Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Implementación de algoritmos de efectos de audio en un procesador DSP de TI

TITULACIÓN: Ingeniería Técnica de Telecomunicación, Especialidad Sistema de Telecomunicación

AUTOR: Cristian Quirante Catalán

DIRECTOR: Gabriel Montoro López

DATA: 06 de marzo de 2008

Título: Implementación de algoritmos de efectos de audio en un procesador DSP de TI

Autor: Cristian Quirante Catalán

Director: Gabriel Montoro López

Fecha: 6 de marzo de 2008

Resumen

Este trabajo quiere dar una idea general sobre los principales efectos utilizados en el audio digital y su confección. Bajo unas explicaciones teóricas previas se pretende llegar a conseguir programar dichos efectos desde su forma más sencilla a construcciones más complejas. Todas las explicaciones teóricas están realizadas antes de las dedicadas específicamente a la programación, ya que será a partir de éstas desde donde nos basaremos para componer dichos efectos. A partir del estudio de los casos reales donde se producen estos efectos llegaremos a las conclusiones de lo que deberemos hacer en el dominio digital para poder simularlos.

La razón porque escogí trabajar alrededor de los efectos de audio viene de mis conocimientos sobre teoría musical y del uso de los propios efectos. Desde hace tiempo utilizo pedaleras de efectos para guitarra y software para de edición musical por lo que estoy familiarizado con el uso, no así con su arquitectura. Y es sobre esto, la arquitectura, lo que se encuentra por debajo, lo que me empujó y me motivó a empezar con este trabajo.

El procesado de los efectos de audio es típicamente en tiempo real por lo que opté utilizar una plataforma capaz de realizar dicho trabajo. Decidí trabajar con una DSP (digital signal processor), más concretamente la C6711 de Texas Instruments, programable en C. Todos los efectos están basados en el procesado muestra a muestra para una posterior utilización en tiempo real. No es así el uso de las muestras de audio, que se toman enteras y no muestra a muestra para ser procesadas. Este hecho disminuye la complejidad, adecuando la carga al tipo de trabajo.

Los aspectos utilizados en detrimento de la complejidad del trabajo quedan como posibles ampliaciones para un proyecto futuro.

El resultado de este trabajo ha sido una pequeña guía acerca del diseño de efectos de audio digital y su base teórica. A nivel más personal espero que la oportunidad de profundizar e indagar sobre todo lo que envuelve este trabajo me empuje a continuar ampliando mis conocimientos y me anime a dedicar tanto futuros estudios como vida profesional.

Title: Algorithm implementation of audio effects with a TI DSP processor

Author: Cristian Quirante Catalán

Director: Gabriel Montoro López

Date: March, 6th 2008

Overview

This project has the aim of giving a general idea about the most common audio effects for digital audio and their composition. Under some previous theoretical explanations we will program those effects from their easiest-way construction until more complex architectures. All the theoretical explanations are done before the ones dedicated to the programming, since we will base on them to compose the effects. From real situations where these effects are produced we will get the conclusions of what we will have to do in the digital domain to simulate them.

The reason why I decided to work around audio effects comes from my knowledge about them. It's been a while since a use pedal effects for guitar and software for audio edition, this is why I'm keen on managing them, but not with their architecture. Is about that, the architecture, what is in the low level, which motivated me to work on it.

Processing audio effects is tipically done in real time so I decided to use a capable platform to do this work. I chose a digital signal processor (DSP), the Texas Instruments C6711 concretely, language C programmable. All the effects are based on sample by sample processing, giving the possibility of a following use in real time. It is not done like that with the audio signals treated, the are entire token, not sample by sample, to be processed. This fact decreases the complexity adapting the load of work.

The aspects used on detriment of complexity are given as possible improvements for a future project .

The result of this project has been a little guide about design and theoretically aspects of digital audio effects. At the personal level I wish that the opportunity of doing this investigation will help me extending my knowledge encouraging me to dedicate future studies or professional life.

ÍNDICE

INTRODUCCIÓN

El objetivo de este trabajo es ofrecer una guía básica del funcionamiento y la programación de los diferentes tipos de efectos utilizados en el audio digital. He elegido los que considero efectos básicos de los cuales hay que tener conocimiento para entender el funcionamiento de las estructuras más complejas, ya que estas estarán siempre basadas en estos efectos básicos.

Como primer capítulo encontrarán todo lo relacionado con las herramientas y dispositivos utilizados para realizar la programación y ejecución, básicamente las DSP TMS320C6711 de Texas instruments y el entorno de programación integrado que ofrece el programa Code Composer Studio. Este capítulo engloba todo lo relacionado con hardware y software.

El segundo capítulo está compuesto por la explicación teórica de los efectos planteados a programar. Cada uno de los efectos supondrá un apartado del capítulo. En cada uno de estos apartados se expondrá de donde surge el efecto y cuales son sus rasgos característicos que hacen obtener las diferentes sonoridades. Todas estas características variables se han intentado simular en la programación de cada efecto para poder experimentar con los diferentes valores que pueden tomar y observar sus peculiaridades. Todas las explicaciones de los diferentes efectos estarán dispuestas dentro de este segundo capítulo. Una vez tomado este primer contacto con cada una de las diferentes estructuras se pasará a explicar la implementación de la programación. Toda lo que envuelve al funcionamiento de la programación estará dentro de este segundo y último capítulo.

En el tercer capítulo se da una visión más técnica del funcionamiento de los efectos, observándolos desde la perspectiva del programador. Para cada uno de los efectos se han desarrollado estructuras que van de las más simples a más complejas. La ordenación está hecha según dicha complejidad, empezando con el ejemplo más simple de cada familia de efectos y acabando con los más complicados. Para una fácil comprensión se ha provisto de una explicación técnica y un diagrama de bloques que define el proceso de cada uno. La última parte importante, también contenida en el tercer capítulo, da una explicación del funcionamiento global de la aplicación. La manera de aplicar los efectos, las limitaciones, el proceso que sigue una muestra o la estructura propia de la aplicación.

Espero que la organización del trabajo consiga ser lo más comprensible y asequible para el lector o consultor.

CAPÍTULO 1. HERRAMIENTAS Y DISPOSITIVOS UTILIZADOS

Este capítulo introduce las diferentes herramientas que he utilizado para el realizar el procesado digital. Estas herramientas incluyen el programa Code Composer Studio, que proporciona un entorno integrado de desarrollo para la segunda herramienta utilizada, la TMS320C6711, una DSP de Texas Instruments. Por otro lado también debo mencionar el uso del simulador, contenido en Code Composer, de la mencionada DSP, que permite hacer pruebas en ausencia de ésta.

1.1. Introducción

Los procesadores digitales como los de la familia de los TMS320C6x están especialmente diseñados con una arquitectura y un set de instrucciones apropiados para el tratamiento de señal. Esta arquitectura, basada en very-long-instruction-word (VLIW), es muy adecuada para cálculos numéricos de gran envergadura.

Este tipo de DSP son ampliamente utilizadas en comunicaciones, audio e imagen dada su gran capacidad y rapidez de cálculo, es este el motivo por el cual la he elegido. Además de por el hecho de que es totalmente reprogramable, pudiendo así ser utilizada para diferentes aplicaciones.

Los procesadores DSP están dedicados primordialmente para aplicaciones en tiempo real. En este trabajo se acabó optando por reducir un poco la complejidad y hacer procesado muestra a muestra enfocado a una aplicación tiempo real pero utilizando señales enteras, no tomadas en tiempo real. Se deja la puerta abierta para una futura continuación/ampliación el conseguir tomar la señal de entrada en tiempo real. Por otro lado el resto del trabajo está enfocado al uso en tiempo real, consiguiendo que el tiempo de procesado de cada muestra sea menor que el tiempo que hay entre muestras, aspecto relacionado con la frecuencia de muestreo de las señales.

El chip de la DSP viene integrado en una placa DSK. La placa tiene todo el hardware necesario para el procesado en tiempo real como un conversor digital para las entradas y salidas. El conversor AD535 que viene en la placa utiliza una tecnología sigma-delta que permite tanto la conversión analógica-digital como la digital-analógica. Un oscilador de 4 MHz, también en al placa, se conecta al conversor para ofrecer una frecuencia de muestreo fija de 8 KHz.

La placa DSK también incluye 16 MB de RAM dinámica y síncrona (SDRAM, synchronous dynamic RAM). También 128 KB de ROM volátil. Dos conectores en la placa ofrecen una entrada y una salida, etiquetadas como IN y OUT respectivamente. El oscilador de la placa es de 150 MHz. Por último, también integrados, están los reguladores de voltaje, uno que da 1.8 V a la DSP y otro que aporta 3.3 V a las memorias y periféricos.

1.2. Procesador digital de señal (DSP) TMS320C6711

La DSP TMS320C6711 está basada en el tipo de arquitectura very-long-instruction-word (VLIW), que es indicada para la realización de algoritmos numéricos de gran carga computacional. La memoria interna programable está estructurada de manera que un total de ocho instrucciones pueden ser ejecutadas cada ciclo. Por ejemplo con oscilador como el que tiene de 150 MHz, la DSP es capaz de llevar a cabo ocho instrucciones de 32 bits cada 1/ (150 MHz) o 6.66 ns. Este oscilador es el que nos dará la cantidad de operaciones a poder realizar entre muestra y muestra, según sea la frecuencia de muestreo de la señal de entrada. Pongamos el ejemplo de que tenemos una señal de audio muestreada a la frecuencia típica de la calidad CD, es decir, a 44.1 KHz. Un ciclo de DSP tiene una duración de 6.6 ns como hemos calculado anteriormente. En la señal de entrada el periodo entre muestras es de 1/44.1 KHz. o 226.75 ms. El cociente entre el periodo entre muestras y el periodo del oscilador de la DSP nos dará el número de ciclos que caben entre una muestra y la siguiente correlativa, (226.75 ms)/(6.6ns), unos 3399 ciclos. Será muy importante conocer este factor a la hora de realizar los algoritmos y calcular que utilicen menos de esta cantidad de ciclos entre muestras, para posteriormente poder utilizarlo en tiempo real.

Los complementos de la TMS320C6711 incluyen 72 KB de memoria interna, ocho unidades funcionales compuestas por seis ALUs y dos unidades multiplicadoras, un bus de direcciones de 32 bits que direccionan 4GB, y dos sets de registros de 32 bits para propósitos generales.

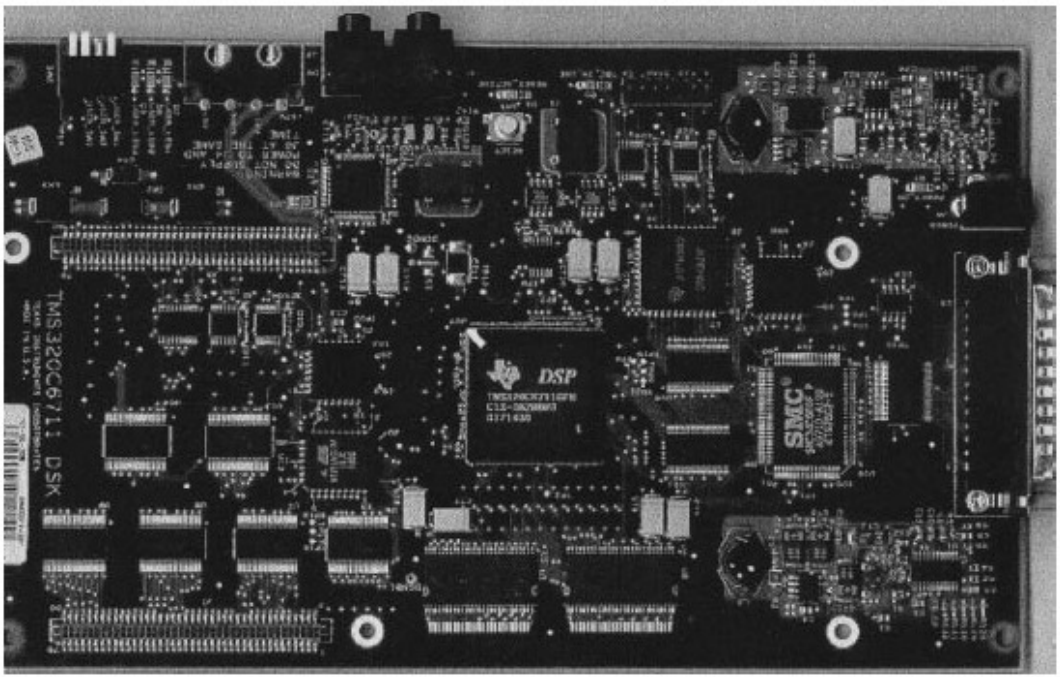


Fig 1.1 Placa DSK con el chip DSP TMS320C6711 incorporado

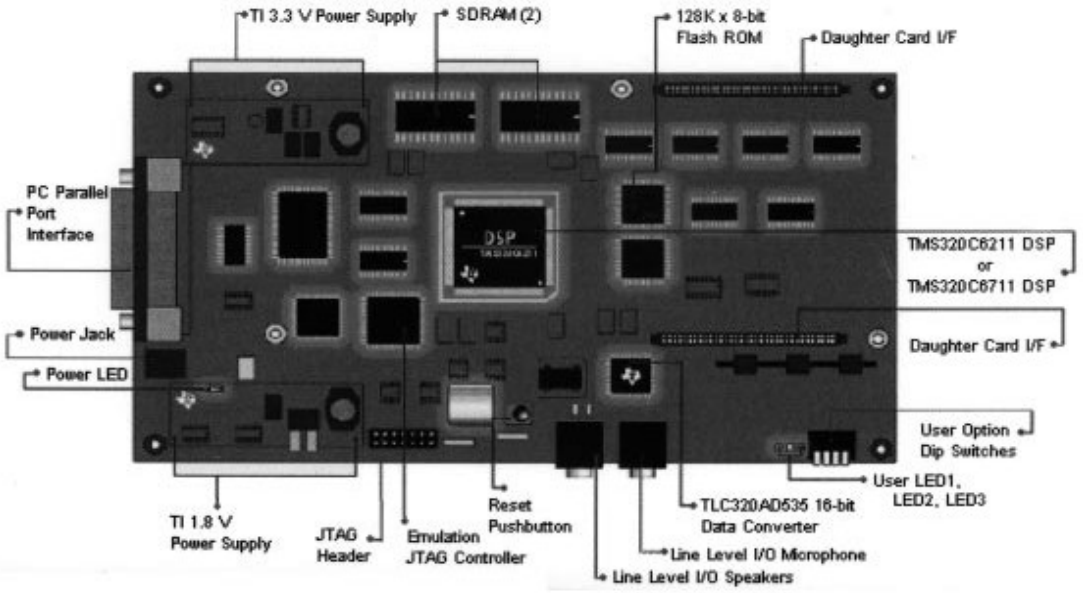


Fig 1.2 Placa DSK vista inferior

1.3. Code Composer Studio (CCS)

Code Composer Studio (CCS) ofrece un entorno de desarrollo integrado (EDI). Incluye herramientas para la generación de código, como un compilador C, un ensamblador, y un linkador. Es capaz de realizar gráficos y soporta el debugado en tiempo real. Dispone de las herramientas de software necesarias para construir y debugar programas que posteriormente podremos cargar sobre la DSP.

El compilador de C compila programas de extensión .c para posteriormente producir un fichero de ensamblaje con extensión .asm. Una vez hecho esto el ensamblador procesa los .asm para producir objetos de lenguaje máquina con extensión .obj. El linkador combina ficheros objeto y librerías objeto como entrada para producir un fichero ejecutable con extensión .out. Este fichero ejecutable es el que finalmente puede ser cargado para ser ejecutado directamente en el procesador digital de señal C6711.

Otras herramientas de nuestro interés dentro de CCS serán las destinadas al testeo de los algoritmos realizados. De esta manera destaco una herramienta de gráficos de las variables que están siendo utilizadas en el programa. Es capaz de realizar gráficos en tiempo y frecuencia y mostrarlos a nuestra conveniencia, marcando los intervalos, los espacios entre muestras, frecuencia de muestreo, etc. Es una herramienta muy útil para analizar las señales de entrada o salida y ver que realmente tiene la forma que deben. Otra herramienta a destacar es el reloj que nos indica los ciclos entre puntos de ruptura. Podemos resetearlo a conveniencia y acumula los ciclos utilizados por la DSP entre dos puntos de ruptura que nosotros hayamos marcado. De esta manera podremos calcular el número de ciclos utilizados en el procesado de una muestra y si se adecua al número de ciclos máximo entre muestras, para realizar el procesado en tiempo real.

CAPÍTULO 2. EFECTOS DE AUDIO DIGITAL

Este capítulo tratará todos los aspectos teóricos que envuelven a los efectos de audio que he tratado durante el trabajo. Cada uno partirá de una explicación previa, para después fragmentar los aspectos variables que definen el sonido característico de los efectos. De este modo encontraremos dos apartados básicos dentro de cada efecto, uno al que he llamado **explicación** y un segundo que recibe el nombre de **variables**.

Por otro lado se han tratado en primera instancia los efectos temporales, en los cuales se trata con las muestras entrantes presentes y otras que ya han sonado con anterioridad. Y en una segunda parte se habla de los efectos de amplitud en los que básicamente se altera la amplitud de la señal presente entrante.

2.1.Efectos temporales

2.1.1.Delays

2.1.1.1.Explicación

Lo que básicamente sucede cuando aplicamos un efecto de delay es que las muestras que van sonando se repiten al cabo de un cierto tiempo de retraso (delay en inglés). Hay muchas maneras de tratar estas muestras pasadas para volver a hacerlas sonar de algún modo junto con las muestras actuales y se pretende dar una visión general de los delays que permita ver de donde se parte y hasta donde se puede llegar con estas líneas de retardo. Los propósitos de un delay pueden ser muy amplios. A partir de un señal seca se puede conseguir que aumente su espacialidad (reverberar), provocar un desvanecimiento de esa señal, crear un caos de reflexiones...Cada tipo de delay tendrá su particular y característico sonido y una gama muy alta de posibilidades con tan solo unas pocas variables. Las variantes que se pueden obtener de una línea de retardo determinada son muchas y aquí se han sintetizado las que pueden dar una visión global de lo que son este tipo de efectos. Además un delay simple es la base para la mayoría de efectos temporales.

2.1.1.2.Variables

Tamaño del delay

El oído humano empieza a captar ecos a partir de una separación entre muestras repetidas a partir de unos 50 ms. En caso de delays inferiores a este

tiempo el cerebro integrará estas y las captará como una especie de cola de sonido que da sensación espacialidad, ya que da la impresión de que el sonido se va perdiendo, como si en una sala confinada estuviéramos. Esta es una manera para provocar reverberaciones y emular el hecho de estar en un lugar donde se producen reflexiones del sonido directo.

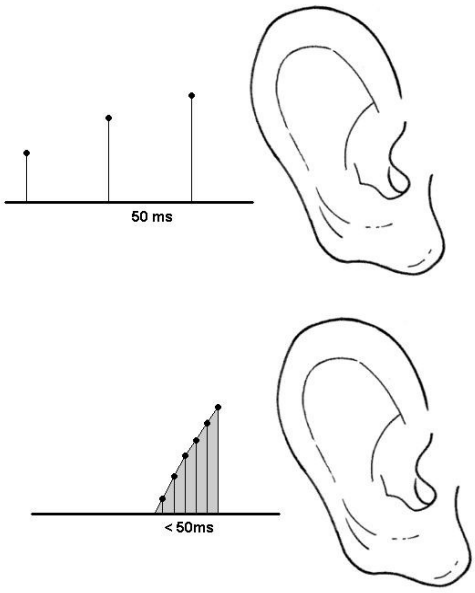


Fig 2.1 Efecto de integración de muestras en repeticiones inferiores a 50 ms

Feedback

Con los delays tenemos la posibilidad de utilizar las muestras retrasadas de entrada o bien crear un sistema con realimentación. En este sistema las salidas serán retrasadas y volverán a sonar un tiempo de delay determinado después sumadas a la señal de entrada. Al cerrar el circuito de esta manera las salidas irán sonando cada tiempo de delay indefinidamente. Estamos pues delante de un sistema tipo IIR con posibilidades de tornarse inestable. El control de este hecho será necesario y deberemos tener una etapa de amplificación / atenuación en la realimentación que llamamos comúnmente *depth*.

Depth del feedback

La realimentación de la salida hacia la entrada producirá que todas estas salidas vuelvan a sonar un tiempo de delay más tarde. Esto comportará que cada una de las muestras seguirá sonando siempre. La clave está en que las muestras que cojamos del buffer para sumarlas a la señal actual las atenuaremos de modo que los sonidos pasados vayan desapareciendo progresivamente. Se irán repitiendo cada tiempo de delay pero cada vez con un volumen menor y se irán perdiendo para el oído, cosa que provocará un efecto de espacialidad, más grande o más pequeño dependiendo de la cantidad de retardo. Si no aplicamos esta atenuación todos los sonidos se irían repitiendo cada tiempo de delay pero

con el mismo volumen siempre, de modo que se irían superponiendo a la señal actual, tomando cada vez más importancia y en poco tiempo la señal sería un caos, dado que el volumen llegaría a la saturación. Este efecto caótico nos puede interesar en algún momento si sabemos y podemos controlarlo, por ejemplo cortándolo antes de que llegue a ese punto de saturación. Si optáramos por no atenuar o amplificar las muestras realimentadas, el sistema se convertiría más tarde o más temprano en inestable. Aun así, en el sonido todo es experimentable, e incluso esta inestabilidad puede tener connotaciones interesantes.

Número de etapas de delay

Aumentar el número de etapas de un delay implica aumentar su complejidad. El número de etapas es equivalente al número de muestras retrasadas con diferentes tiempos de delay que sumamos a la muestra actual. En un esquema con más de una etapa de delay podemos también tener una realimentación de la salida, que vaya a todas las etapas, o también podemos obtener esquemas más complicados haciendo que las realimentaciones sean varias y vayan en múltiples direcciones. Todas estas realimentaciones serán controladas por las variables que comentábamos anteriormente, sus correspondientes depths. Es decir, podrán haber tantos depths como etapas tenga la línea de delay.

Número de canales

El número de canales también puede resultar interesante a la hora de construir una línea de delay. Si por ejemplo tratamos con una señal estereo podemos actuar sobre cada canal independientemente del otro, podemos cruzarlos, aplicar distintos delays a cada uno, etc. Las posibilidades se pueden disparar si tratamos con señales de sonido envolvente de 5 o incluso 7 canales.

2.1.2.Chorus

2.1.2.1.Explicación

Chorus, es la traducción de coro, en inglés. Un coro se compone de un grupo de cantantes, así pues, el efecto de chorus se referirá a algo similar. El efecto consiste en que la señal que suena, sea sumada a ella misma un número *n* de veces. Pero esta suma no es tan sencilla. Si añadiésemos una señal encima de la otra, al ser exactamente iguales lo único que estaríamos haciendo es sumar frecuencias exactamente iguales, y por tanto se produciría un aumento de volumen. Lo que ocurre es algo más complicado. Para entenderlo mejor utilizaremos un ejemplo concreto, pongamos dos o más guitarristas. Tenemos que analizar que se produce cuando estos tocan al unísono la misma frase o acorde. Nos debemos preguntar, ¿por qué en ese caso suena un efecto de chorus y no se produce un simple aumento de volumen? Los guitarristas intentarán tocar las notas en los mismos instantes de tiempo, pero la imprecisión humana no se lo permite. De modo que habrá un desfase entre las señales que suenan a la vez. Además este desfase será aleatorio porque en ocasiones será uno de los intérpretes el que toque antes la nota y en otro momento seguramente la toque más tarde, ya que los desfases son de pequeña magnitud. Además de este hecho hay otro determinante y es que la afinación no es exacta, además las cuerdas no son exactamente las mismas, ni el instrumento en sí tampoco. Todo ello conlleva que el tono que produce cada músico es un poco distinto, hay una leve desafinación relativa entre ellos, que también resulta aleatoria. Estos factores son los que principalmente producen el chorus. Otro factor que también se produce, pero tiene una importancia menor, es que los músicos no siempre tocan con la misma fuerza el instrumento, y cada nota suena con una determinada potencia/volumen. Este hecho también es aleatorio y se puede simular, aunque no sea tan necesario. Después este efecto puede tener infinitos matices distintos, según el número de instrumentos tocando a la vez, el grado desafinación entre ellos, el desfase y la diferencia de volúmenes.

El desfase temporal, se puede producir mediante una línea de delay. Pero un delay simple no produciría la aleatoriedad del desfase. Sonarían las señales siempre desfasadas un tiempo determinado. Para producir esta imprecisión podemos hacer un delay de un tiempo variable, y que el intervalo de cambio sea pequeño, tal y como pasaría con dos músicos tocando a la vez. Esto lo conseguimos con un oscilador que vaya cambiando la cantidad de delay. Pero se produce un problema si ponemos un oscilador de una frecuencia demasiada alta, ya que la señal se va comprimiendo y expandiendo a demasiada velocidad y esto produce un cambio en el tono que puede resultar poco adecuado. El problema lo resolvemos poniendo un oscilador de baja frecuencia, porqué además sigue produciendo una compresión/expansión de la señal en el tiempo, pero al ser más lenta produce un leve cambio en el tono, muy similar al que se puede producir en la realidad cuando observamos un coro. Por lo tanto, se puede decir que matamos dos pájaros de un tiro, el desfase por un lado, y la desafinación por otro. Si quisiéramos obtener también el volumen aleatorio de cada nota podíamos tener un oscilador de amplitud no constante.

2.1.2.2.Variables

Tiempo de delay

El delay característico de un chorus actúa en un intervalo de entre 20 ms y 30 ms. Por lo tanto debemos distinguir entre una parte fija, que serían los 20 ms, y una parte variable, que serían los 10 ms, que hay de 20 ms a 30 ms. Por encima estos tiempos el oído ya empezaría a notar el desfase más como un delay que como un chorus. La velocidad con que cambie el delay variable determina el cambio de tono que sufra la señal. El siguiente esquema resulta bastante aclaratorio:

Fig 2.2 Variación del tamaño de delay en un chorus

El oscilador de baja frecuencia o LFO (Low frequency oscillator)

Normalmente el LFO se sitúa en frecuencias menores a 3 Hz. Se fija una frecuencia así de baja para conseguir lo que mencionábamos en la definición del efecto, que la señal sumada a la original no se comprima y se expanda tan rápidamente que conlleve una desafinación muy grande. De cualquier forma esto es orientativo ya que podemos obtener efectos más radicales aumentando esta frecuencia, todo dependerá del sonido que queramos obtener, más clásico o experimental.

Fig 2.3 Control del delay variable mediante un LFO

Tipos de ondas para el LFO

Dependiendo del tipo de onda, variará de una u otra manera la línea de delay. Cuando el retraso se encuentra aumentando la señal se expande, y por tanto, el tono se torna más grave. Cuando llega al máximo el retraso también lo es. Y cuando el retraso decrece, sucede el efecto contrario, el tono se vuelve un poco más agudo ya que la señal se comprime. Cuanto más estrechas (rápidas, más frecuentes) sean las oscilaciones del LFO más variará el tono de la señal modificada, ya que la señal se expandirá o contraerá rápidamente.

Las ondas más comunes que podemos utilizar son la sinusoidal, triangular o logarítmica, de las cuales podemos variar sus características: frecuencia y amplitud. De esta manera obtendremos efectos distintos, que nos pueden resultar más o menos adecuados, según lo que busquemos.

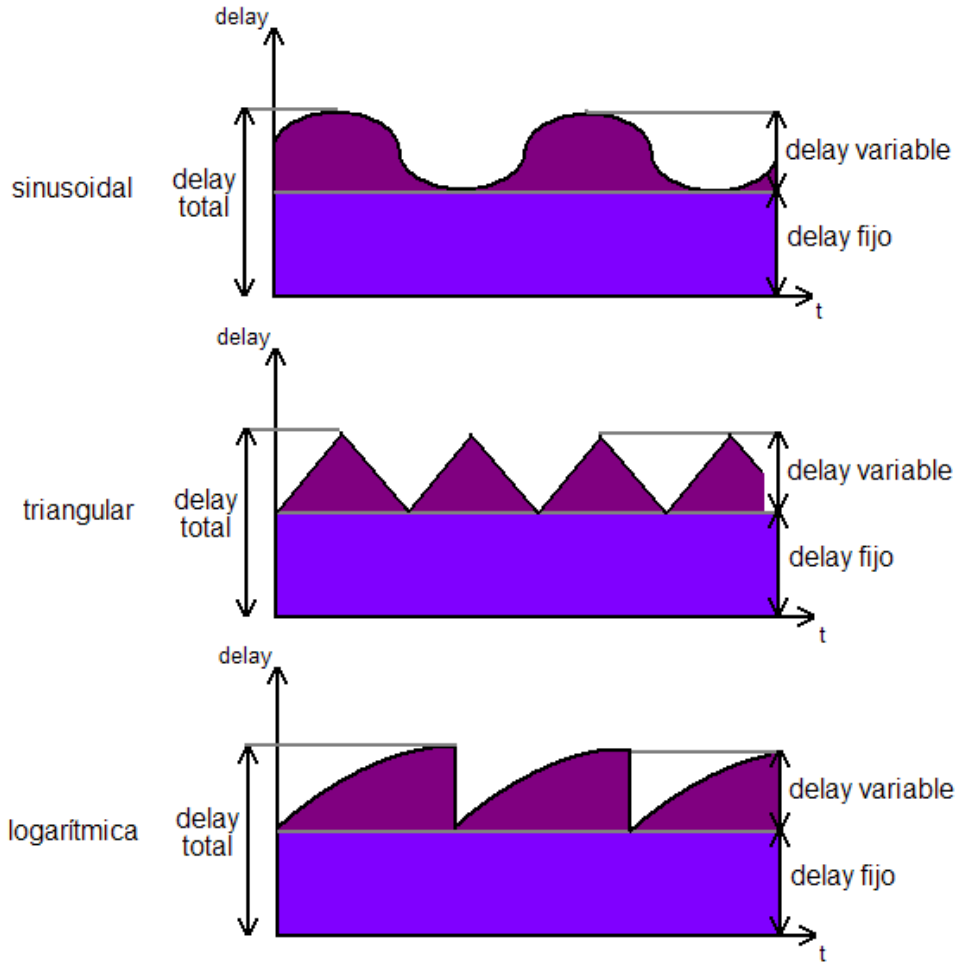


Fig 2.4 Tipos de ondas generadoras del tamaño de delay

Feedback

El chorus coge muestras retrasadas de la señal para ser conformado. Podemos elegir almacenar las muestras de entrada (sin feedback) o bien guardar la salida procesada por el efecto. En caso de que escojamos reutilizar las muestras de entrada obtendremos un chorus sencillo que es el que se produce en el mundo real cuando varios instrumentos tocan al unísono. Si escogemos realimentar el sistema y reutilizar las muestras de salida conseguiremos más intensidad en el efecto ya que las muestras serán procesadas por el chorus repetidas veces.

Múltiples voces

Como en la vida real, para realizar un coro podemos hacer uso de dos o más instrumentos dependiendo de lo que pretendamos. La simulación de este hecho consiste en tomar más de una muestra retrasada utilizando varios LFOs situados en distinta fase, con distinta forma de onda, de mayor o menor frecuencia, etc. De esta manera conseguiremos distintos tiempos de delay para las distintas muestras retrasadas. Podemos simular tantas voces como queramos atendiendo a que conforme añadamos el grado de complejidad de computación también irá aumentando. Evidentemente el grado de intensidad irá en aumento a medida que tengamos más voces en el efecto y aun más sin añadimos un feedback para todas esta muestras retrasadas. Podemos conseguir gran cantidad de efectos en este sentido.

2.1.3.Flanger

2.1.3.1.Explicación

El efecto de un flanger es conocido por tener una sonoridad similar a la que genera una turbina de un avión en la distancia. El flanger actúa como una especie de filtro que afecta a la respuesta frecuencial de la señal, provocando una serie de ceros a lo largo del espectro. Es este suceso es el que básicamente provoca la sonoridad del flanger.

Es un efecto que se compone de manera similar a un chorus. Para conseguir el efecto del flanger bastaría con sumar a la señal de entrada ella misma con un cierto retardo. Este delay debería de estar entre 1 y 10 ms. El oído humano no captará este retardo como un eco, sino como un efecto añadido a la señal, lo que llamamos flanger. La diferencia con un simple efecto de delay radica en el tiempo que tiene el retardo. Hasta unos 70 ms el oído no capta ningún eco como ya se ha comentado en algún apartado. Es importante también colocar un control de volumen en la rama de la señal añadida, comúnmente llamado depth.

Como comentábamos al principio, lo que en realidad está produciendo el delay es un filtrado de la señal. La respuesta frecuencial cambia y se producen una serie de ceros en frecuencia (notches), que hacen eliminar ciertas componentes frecuenciales y las demás sufren un cierto cambio de amplitud.

Imaginemos que tenemos nuestra señal como el figura 2.5:

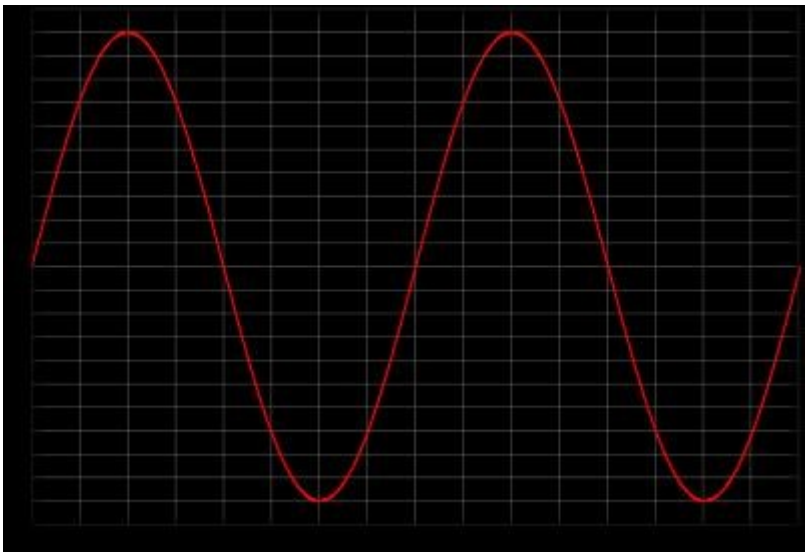


Fig 2.5 Onda senoidal considerada sin desfase

Si le sumamos ella misma con un retardo que haga coincidir los mínimos con los máximos, desfasada exactamente medio ciclo:

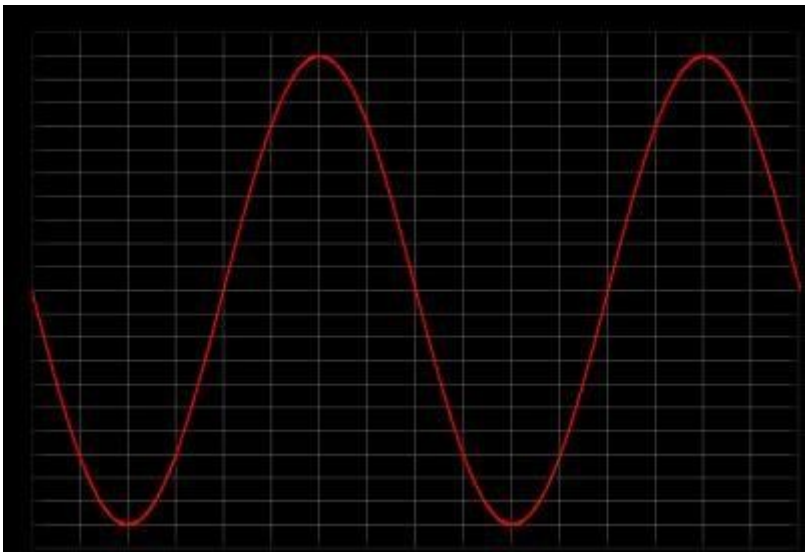


Fig 2.6 Onda senoidal desfasada medio ciclo respecto a la figura 2.5

La señal quedará anulada completamente. Se crea una interferencia destructiva. Fijémonos que se trata de un tono perfecto, por lo tanto si esto sucediera en la realidad, esta frecuencia en concreto quedaría eliminada. Para que esto sucediera deberíamos tener un depth de 1. Lo que normalmente sucede es que el depth es menor a 1, en 0 la respuesta frecuencial sería plana, conforme nos vamos acercando a uno los ceros o notches son más profundos. Pero aunque las componentes frecuenciales nos sean completamente anuladas si que quedan atenuadas, además, las bandas que rodean estas componentes también quedan atenuadas. Cuanto más cercanas se encuentran al notch más reducidas quedan. El efecto que ello produce, aunque no sea una situación ideal, es perfectamente audible.

Para obtener un flanger más complejo y con un sonido más característico debemos hacer que los notches vayan cambiando a lo largo del tiempo. Para conseguirlo tenemos que actuar sobre el tamaño del delay. Cuanto más grande sea el delay los notches se irán hacia las frecuencias bajas. Pongamos que tenemos un tono de 500 Hz, y le sumamos la misma señal retardada para cancelarlo. Si reducimos la frecuencia del tono a 250 Hz y retardamos la señal con un tamaño de delay igual que en el anterior ejemplo, observamos como no se produce la cancelación de esta componente. Si queremos eliminar una frecuencia más baja deberemos aumentar el tamaño de delay.

Veamos que relación hay entre el tamaño de delay y la frecuencia concreta atenuada. Si queremos eliminar un tono de una frecuencia de por ejemplo 2 KHz debemos tener una señal retrasada medio ciclo. Un tono de 2 KHz tarda en hacer un ciclo 1 / 2 KHz = 0,5 ms. Por tanto si queremos retrasar la señal medio ciclo necesitaremos un delay de la mitad de 0,5 ms, es decir, 0,25 ms. Por lo tanto concluimos que si queremos que nuestro primer notch se encuentre en 2 KHz debemos tener un tamaño de delay de 0,25 ms. El siguiente notch se encontrará 1 / 0,25 ms después, es decir a 6 KHz.

Si tomamos un ejemplo al contrario y tenemos un delay de 2 ms, ¿dónde caería ahora el primer notch? 2 ms tiene que ser la mitad de lo que dura un ciclo de la frecuencia que queremos eliminar. De manera que la componente a eliminar debe tener un ciclo de duración 4 ms. Para determinar que frecuencia corresponde a un periodo de 4 ms tan solo debemos hacer su inversa, 1 / 4 ms = 250 Hz. La siguiente componente estaría 1 / 2 ms después, es decir 500 Hz más arriba, a 750 Hz, y así cada 500 Hz.

2.1.3.2. Variables

Delay dinámico

Para obtener este cambio de tamaño de delay deberemos utilizar un LFO que vaya actuando sobre la cantidad de retardo, desde un mínimo a un máximo. Cuando el LFO esté en su máximo el retardo también lo estará, y sucederá lo contrario cuando se encuentre en un mínimo. En cuanto a la frecuencia del LFO dependerá de cuanto queremos que tarde en subir y bajar los notches desde su frecuencia más baja a la más alta. De este modo, con un LFO de 3 Hz la subida y bajada se repetirán 3 veces en un segundo. Un valor típico para el LFO sería entre 0.5 Hz y 3 Hz.

Una última consecuencia de ampliar o reducir el delay continuamente es que se produce un cambio en el pitch de la señal constantemente. Esto quiere decir que la señal retrasada sonará un poco más aguda o un poco más grave, según el momento. Si nuestro LFO está en subida, acercándose a un máximo el retraso cada vez será mayor y la señal se tornará un poco más grave. Sucederá lo contrario si se acerca a un mínimo.

Volumen de la señal retrasada (depth)

Al igual que sucedía con los delays y chorus, el depth es el control que ponemos a la salida del delay del flanger. Nos permitía hacer que los notches fueran más o menos profundos. Si aumentamos el depth obtendremos ceros más pronunciados, si lo reducimos, las atenuaciones serán más suaves.

Tamaño del delay

Como ya hemos dicho debe encontrarse entre 1 ms y 10 ms. Conforme más grande hagamos el delay más cercanos a las frecuencias bajas estarán los notches. El delay se moverá entre un valor mínimo y un máximo. Si este baja de 1 ms el efecto de flanger desparecerá al oído humano momentáneamente hasta que el retraso vuelva a ser superior a 1 ms. En caso de superar los 10 ms el efecto que se empezaría a notar sería más similar al de un chorus.

Feedback

Es una opción que se puede adoptar para conseguir un sonido más experimental, similar al producido como si estuvieras dentro de un tubo de metal. Puedes tomar la salida del delay y volverla a sumar a la entrada (realimentación o feedback) y descubrir las posibilidades que te proporciona esta nueva rama. Recordar que esta realimentación se tiene que atenuar con un valor hasta 1, de otra manera el sistema podría volverse inestable.

2.1.4.Phaser

2.1.4.1.Explicación

El phaser es un efecto que se produce cuando a la señal de entrada le cancelamos algunas componentes frecuenciales, y variamos estas frecuencias anuladas con el tiempo. Tiene relación, como su nombre indica con la fase de la señal. Para llevarlo a cabo le sumamos a la señal entrante ella misma filtrada a través de un filtro paso todo de características especiales. Aquí es cuando nos formulamos la pregunta de que si se trata de un filtro pasa todo, ¿no estaremos sumando simplemente dos señales iguales? Ciertamente, un filtro pasa todo lo que hace es mantener la misma amplitud para todas las frecuencias. Es decir el comportamiento de la amplitud con respecto a la frecuencia es plano. No siendo así en el caso de la fase, ya que esta podemos hacer que sea muy oscilante en función de la frecuencia. Este será el factor determinante que nos permitirá variar en el modo que queremos la señal.

Veamos primero de todo, para que sea más comprensible, el comportamiento de un filtro pasa todo cualquiera. Utilizaremos la herramienta Matlab para crear un filtro y obtener su respuesta frecuencial de amplitud y fase.

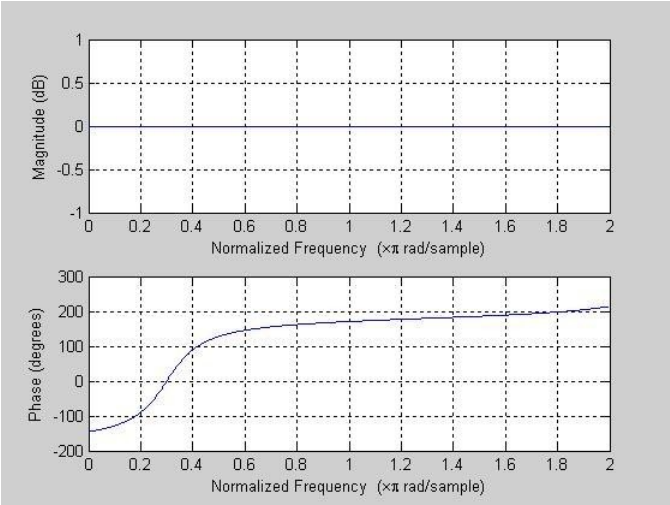


Fig 2.7 Respuesta frecuencial en amplitud y fase de un filtro pasa todo

En la parte superior tenemos la respuesta en amplitud y justo debajo su respuesta respecto a la fase. Como se puede observar el eje frecuencial está normalizado a la frecuencia de muestreo que se utilice. La fase la tenemos anotada en grados. Como avanzábamos en el apartado anterior notamos la respuesta plana de la amplitud y la no lineal de la fase.

El retraso en grados de una cierta frecuencia se explica mediante una onda periódica de frecuencia pura cualquiera (una sinusoide, una onda triangular, etc.). La distancia entre dos puntos iguales de una onda periódica se define por una distancia de 2π radianes, que es equivalente a una vuelta entera, es decir

360º.

Es decir, un retraso de π radianes equivaldrá a un retraso en grados de 180º y así con el resto de fracciones de ángulos. Todo esto lo venimos a explicar ya que se produce un hecho determinante con ciertos retrasos. El caso concreto es el de los retrasos de 180º o -180º, o cualquier ángulo que sea 180*n donde n es un número impar, ya sea positivo o negativo. Una explicación detallada del porque se eliminan una ciertas frecuencias, queda detallado en el apartado anterior de la explicación del flanger.

El comportamiento del filtro pasa todo en este caso es igual que el comportamiento de un filtro notch, que elimina frecuencias concretas. De modo que según sea más o menos complicada la respuesta frecuencial de la fase de nuestro filtro, podremos tener más o menos frecuencias retrasadas 180º y por tanto canceladas. Se puede observar también que las frecuencias que rodean a estas canceladas también quedarán de alguna manera atenuadas. Cuanto más cercanas estén a la frecuencia anulada, más reducidas quedarán, ya que tendrán retrasos muy similares a 180º y por tanto serán componentes muy parecidas a las de la señal original pero de signo contrario.

Al mover en el tiempo estas frecuencias canceladas, cambiando la composición del filtro, es cuando obtenemos el sonido deseado, característico del phaser.

2.1.4.2.Variables

Tipo de filtro

Las respuesta de la fase de un filtro a otro puede variar mucho, según su orden sobretodo. Los filtros de primer orden, los más sencillos, tienden a cancelar frecuencias altas del espectro. Si queremos que estas frecuencias bajen a un espectro más audible y que la cancelación no sea única deberemos aplicar filtros de mayor orden. Para obtener un filtro de mayor orden podemos simplemente poner dos o más filtros en cascada.

Número de frecuencias canceladas

La respuesta frecuencial de la fase nos dictará el número de paso por 180º que esta tiene. Cada paso corresponderá a una componente frecuencial anulada. Cuanto más complejo sea el filtro más frecuencias será capaz de cancelar.

Ancho de frecuencias canceladas

Cuanto más plana sea la respuesta frecuencial de la fase en un paso por 180º más ancho será el espectro de frecuencias anuladas o atenuadas por ese paso. Si tuviéramos un filtro cuya respuesta frecuencial fuera plana y en 180º no tendríamos otra cosa la señal original negativa, por lo tanto si la sumásemos

a la original obtendríamos una salida nula.

Número de filtros

Se pueden sumar cuantas copias de la señal entrante se deseen, y ser filtradas por filtros pasa todo distintos o con parámetros cambiados, sumándose al final. Además cada etapa de filtrado puede contener uno o más filtros en cascada. Esto implica que las estructuras se pueden volver muy complicadas.

Tiempo de ascenso y descenso de las frecuencias canceladas

Para obtener el efecto distintivo del phaser debemos dinamizar la respuesta frecuencial de la fase de nuestros filtros. De esta manera, debemos hacer subir y bajar las frecuencias canceladas empezando y acabando en un cierto punto.

Máxima y mínima frecuencia cancelada

Son los puntos frontera desde y hacia donde se van a desplazar las frecuencias canceladas.

2.2.Efectos de amplitud

2.2.1.Trémolo

2.2.1.1.Explicación

El efecto de trémolo consiste en unas bajadas y subidas en la amplitud de la señal coordinadas con el tempo de esta. En principio deben estar coordinadas porque típicamente se ha hecho así, aunque siempre hay cabida a cualquier tipo de experimentación. Esta sincronización se traduce en que en un compás haya un número entero de subidas y bajadas.

Para obtener este efecto lo que hacemos es multiplicar la señal de entrada por una onda periódica, normalmente una sinusoidal. En términos de modulaciones esto es lo que se entiende por una modulación de amplitud (AM). En la figura 2.8 tenemos un ejemplo, el primer gráfico corresponde a la señal de entrada, el segundo a la onda moduladora y el tercera al resultado de la modulación.

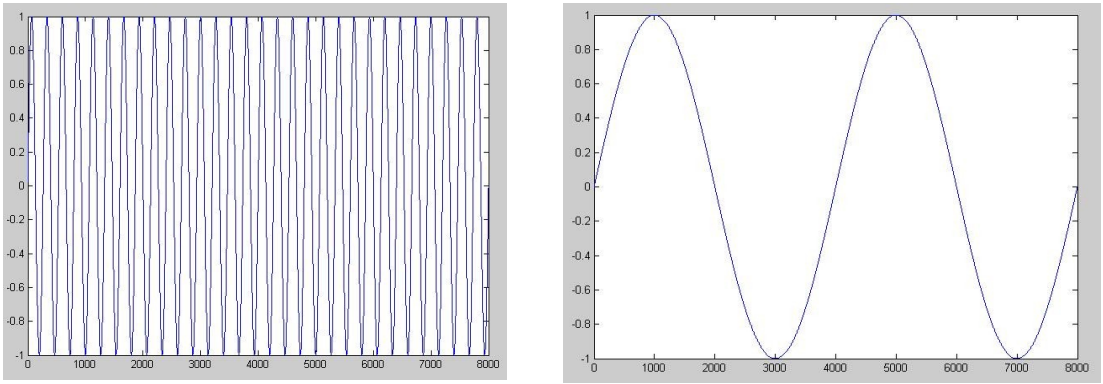
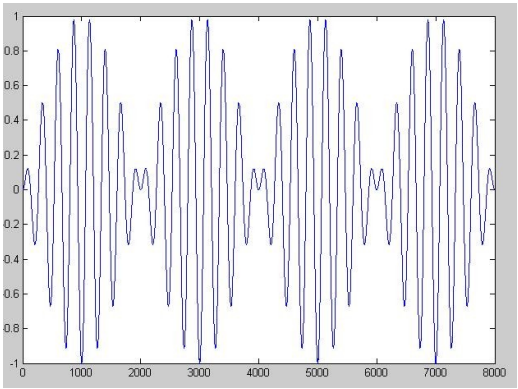


Fig 2.8 Equivalencia del efecto de tremolo con una modulación AM



Podemos observar como la onda sigue manteniendo su frecuencia pero con unos altibajos provocados por la onda moduladora de baja frecuencia que proporciona la forma a la envolvente.

Para calcular que frecuencia debe tener la onda que multiplicamos a la señal de entrada para que esté coordinada debemos realizar unos sencillos cálculos. Supongamos que tenemos una señal de entrada que tiene un tempo de 120 bpm (beats per minute) y que tiene un ritmo 4x4.

Nº compases = 120 bpm / 4 beats por compás = 30 compases por minuto **(2.1)**

(2) Duración compás = 60s / 30 compases = 2 s **(2.2)**

Para que exista coordinación si un compás tiene 2 s, el periodo o periodos de una onda nos tienen que caber justos en ese tiempo. Para calcular la frecuencia que necesitaremos para producir el efecto deseado tenemos que fijar nosotros cuantas subidas y bajadas queremos en un compás.

(3) f_{onda} = nºperiodos / duración compás **(2.3)**

Si por ejemplo queremos que haya 4 periodos en un compás de nuestro ejemplo:

(4) f_{onda} = 4 periodos / 2 s = 2 Hz **(2.4)**

2.2.1.2.Variables

Tipo de onda periódica

Utilizaremos distintas formas de onda según queramos que las bajadas o subidas de la amplitud sean más o menos bruscas. Si por ejemplo utilizamos una onda cuadrada los cambios serán muy bruscos. Si utilizamos un diente de sierra la bajada o la subida será más abrupta. Con una triangular o sinusoidal tendremos cambios más suaves.

Periodo de la onda en un compás

Como ya hemos comentado es importante que los periodos de la onda que utilicemos quepan justos en un compás para obtener la sensación de sincronía entre el efecto y la señal.

Diferentes periodos en un compás

Podemos utilizar varias ondas con diferentes periodos para obtener distintas sonoridades y peculiaridades. Una técnica podría ser utilizar ondas con diferentes periodos, múltiplos unos de otros en un mismo compás, por ejemplo periodos 2*n. Obtendríamos unas bajadas el doble o la mitad de rápidas o lentas. La fórmula compacta para calcular la frecuencia de la onda que deseamos se expresa de la siguiente manera (en el caso que tengamos un ritmo 4x4) :

$$f_{onda} = np / (60 / (tp / 4)) \quad \textbf{(2.5)}$$

f_{onda} = frecuencia de la onda deseada.
np = nº periodos que queremos dentro de un compás.
tp = tempo

2.2.2.Compresor

2.2.2.1.Explicación

Un compresor actúa sobre la dinámica de la señal. De alguna manera equilibra los niveles de la señal. Si una señal tiene niveles de amplitud muy distintos hace que los niveles más bajos ganen presencia y que los altos no acaparen toda nuestra atención. Digamos que amplifica los niveles más bajos y atenúa los de más volumen, todo en una cierta medida.

La mejor manera de entender un compresor es mediante su respuesta de entrada/salida. Si la entrada es menor que un cierto nivel el compresor le aplicará una ganancia para que la señal gane en presencia. En caso de que se mantenga en un nivel medio el compresor no actúa y deja que la salida tenga el mismo nivel que la entrada. Si la entrada supera un cierto nivel marcado el compresor vuelve a actuar y atenúa, reduce la señal evitando que tenga demasiada presencia. Estos hechos se pueden modelar con una función de entrada salida mediante rectas con distinta pendiente para cada nivel.

Fig 2.9 Respuesta nivel entrada/salida de un compresor

Se puede utilizar en muchos aspectos. Puede servir para sostener más tiempo la nota de un cierto instrumento, por ejemplo las notas de un piano. Si aplicamos un compresor a la señal salida de un piano cuando toquemos una nota y esta empiece a decaer el compresor amplificará un poco cuando empiece a tener un nivel bajo y hará que la caída sea más suave, sostendrá la nota más tiempo. También se puede utilizar durante la producción para mantener una voz o un instrumento dentro de un margen dinámico adecuado. Un músico puede tocar una pieza casi a la perfección, pero puede que se mueva mientras está sentado o de pie y se acerque o aleje demasiado del micrófono, o puede que alguna nota la toque con demasiado poco énfasis. Entonces entra en acción el compresor y ajusta los niveles haciendo que toda

la señal captada suene a un nivel más adecuado, que no haya demasiados bajadas o subidas inesperadas de amplitud.

Otro uso podría ser una vez hayamos entrado en la mezcla final de una producción. Cuando tenemos la mezcla final de una canción puede que haya instrumentos o sonidos que no tengan la presencia que deseamos. Por ejemplo un *charles* de una batería que suena muy por debajo y hace que el ritmo no sea tan constante como deseamos. Aplicando un compresor, como hemos visto, podemos amplificar este tipo de sonidos de bajo nivel y hacerlos más presentes en la mezcla final. Siempre se aplica un compresor al final de la mezcla, 'todo se escucha más'. En ocasiones puede parecer que estamos aumentando el volumen general pero en realidad no es así, todo se escucha con más cuerpo por la subida de volumen inteligente en las partes que así lo necesitan.

2.2.2.2.Variables

Límite de niveles bajos y altos

Para caracterizar nuestro compresor debemos definir lo que nosotros vamos a considerar un nivel bajo y alto. Una vez hecho esto tendremos los tres rangos, el de niveles a amplificar, los niveles a dejar pasar tal cual y los niveles a atenuar.

Amplificación de los niveles con menos presencia

Es la amplificación que daremos a aquellos niveles de señal los cuales no percibimos adecuadamente y creemos que deben aumentar su presencia.

Atenuación de los niveles con más presencia

Se utiliza para compensar el hecho de que tengan demasiada presencia. De esta manera los niveles bajos no lo son tanto relativamente con los altos y todo 'se escucha más'.

2.2.2.3.Tipos de compresor

Según los valores que seleccionemos para caracterizar nuestro compresor obtendremos respuestas propias de los diferentes tipos de compresor existentes. Aquí describiremos dos de las posibilidades la puerta de ruido, noise gate en inglés, y el limitador.

Puerta de ruido o noise gate

Actúa de manera contraria al compresor , pero el funcionamiento es muy similar. Hace efecto sobre las partes de la señal con menos nivel. A partir de un cierto límite, atenúa mucho la señal para que sea menos perceptible o incluso imperceptible. Podía esto llevarnos a pensar que el resultado es antagónico al del compresor donde los sonido de menor presencia, pasarían a no tenerla directamente. Pero el caso es que se utilizan en situaciones completamente distintas. La puerta de ruido se utiliza en su mayoría para tratar el ruido de fondo de alguna señal. Por ejemplo el ruido producido por un amplificador de guitarra que proviene del ruido de la red eléctrica. Es un sonido molesto de fondo que tiene menos nivel que la señal tocada por el instrumento. Aplicando correctamente el punto límite donde empezar a atenuar los niveles bajos podemos hacer que tenga aun menos importancia reduciéndolo. Mediante una curva parecida a la respuesta entrada/salida del compresor, podemos explicar el funcionamiento de este efecto que cambia la dinámica de la señal.

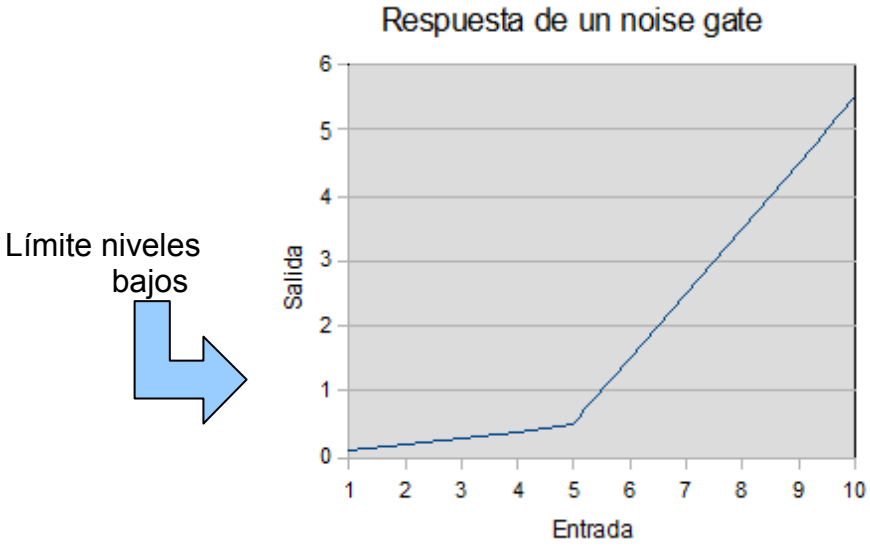


Fig 2.10 Respuesta nivel entrada/salida de un noise gate (puerta de ruido)

En este caso particular para una entrada relativa de hasta 5 las entradas quedarían muy atenuadas mientras que tendrían una salida 1:1 a partir de este límite.

Limitador

Se utiliza para que pese a tener golpes de volumen elevados, estos no salgan con una potencia elevada y queden 'limitados'. Para conseguir este efecto atacamos a la dinámica de la señal atenuándola fuertemente cuando ésta supere un cierto nivel que consideremos alto. De esta manera la respuesta entrada / salida de un limitador quedaría caracterizada de la siguiente manera.

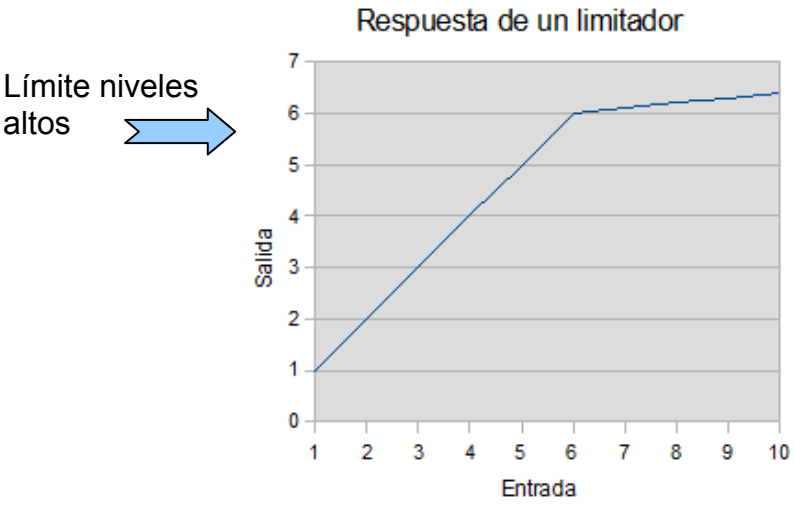


Fig 2.11 Respuesta nivel entrada/salida de un limitador

2.2.3.Distorsión

2.2.3.1.Explicación

Hay muchos tipos de distorsión pero la que se ha tratado en este trabajo ha sido la que tiene relación con la amplitud de la señal. Las primeras distorsiones que se encuentran en audio provienen de los años 60 cuando guitarristas subían mucho el volumen de sus amplificadores, cosa que colapsaba el margen dinámico y se producían, hasta la fecha, sonidos indeseados. Muchos músicos de la época supieron sacarle partido y obtener melodía a partir de efectos en principio no deseados y para los cuales no estaban diseñados los amplificadores. En estos términos tenemos el nacimiento de las primeras distorsiones, voluntarias claro. A partir de entonces los diseños de amplificadores y pedales han avanzado mucho y podemos encontrar multitud de tipos de distorsionadores, utilizados en diferentes ámbitos de la música, y no solo enfocados a las guitarras. La música electrónica tiene gran parte de culpa, con su sintetizadores y bajos distorsionados para la obtención de los típicos sonidos denominados como 'ácidos'.

2.2.3.2.Variables

Lo que en este proyecto se ha tratado de emular son esas primeras distorsiones que intentaban colapsar el margen dinámico y así aplanar la señal a partir de ciertos volumen que el mismo amplificador no era capaz de sacar.

Límite de volumen

El límite de volumen podrá variar desde 0 hasta el tamaño máximo del margen dinámico de nuestro sistema. Cuanto más lo reduzcamos mayor impacto haremos sobre la señal y mayor distorsión sufrirá. Conforme lo aumentemos la distorsión será más suave y el impacto será menor. De esta manera podemos elegir darle un aspecto más duro y contundente o un simple toque a la señal. En la figura 2.12 en el primer gráfico observamos un tono distorsionado levemente, en el segundo un distorsión mas dura sobre el mismo tono.

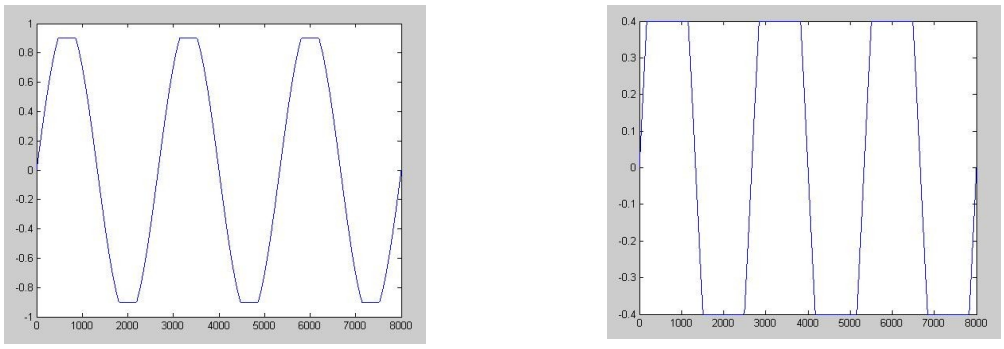


Fig 2.12 Tono con dos tipos de distorsión

CAPTITULO 3.FUNCIONAMIENTO E IMPLEMENTACIÓN

3.1.Explicación técnica

El trabajo parte de la programación por separado de cada uno de los efectos. Cada uno se establece como una función, independientes las unas de las otras. Todos las funciones que realizan los efectos pertenecientes a una familia (delays, chorus, flangers, etc.) están ordenadas y dispuestas en un fichero .c . Cada fichero del que hablamos alberga el efecto más simple posible de la familia, el que podríamos llamar efecto raíz, y a partir de éste una serie de variantes que difieren en complejidad y características, que tienen en común el basarse en ese que llamamos efecto raíz (Ejemplo: Efecto raíz delay simple, variante delay con realimentación).

A parte de los ficheros con las familias de efectos tenemos otro que contiene las funciones generales que no producen efectos, como por ejemplo las encargadas de leer los ficheros de sonido o escribirlos. Y otro tipo de fichero, uno de cabecera concretamente, donde aparecen las definiciones de las variables definidas por el usuario y las cabeceras de todas las funciones.

Existe por último un fichero 'main' que es el encargado de la ejecución. En éste se determinan las características concretas de cada efecto, cuales serán los efectos utilizados, cuanto tiempo de señal será procesada, y el resto de variantes que iremos viendo y explicando.

3.1.1.Procesado muestra a muestra

Todas las funciones están programadas de tal manera que lo que reciben es una muestra de entrada, los parámetros que definen el efecto, y en algunos casos el buffer o buffers que se encargan de almacenar las muestras, y los índices que gestionan en qué posición se encuentran estos buffers. Las funciones tras recibir todas estos parámetros elaboran una muestra de salida. Es decir cada función la podemos ver como una caja negra donde entra una muestra y sale otra nueva muestra procesada. Por tanto, estamos delante de un sistema que utiliza un procesado muestra a muestra.

3.1.2.Procesado en tiempo real

El procesado en tiempo real es el que utilizan los sistemas que deben tener retardos negligibles al oído entre la entrada y la salida. En el periodo que transcurre entre que la muestra entra y sale, que el inverso a la frecuencia de muestreo de la señal, tenemos que realizar todas las operaciones/acciones necesarias. En caso de que el procesado tuviera una envergadura demasiado grande como para ser procesado con nuestra DSP tendríamos problemas a la hora de sacar la muestra en el momento adecuado. El proyecto está preparado para este tipo de procesado. En una posible ampliación uno de los objetivos

sería adecuar la señal de entrada muestra a muestra y obtener una salida en tiempo real.

3.1.3.Almacenamiento de muestras.

Muchos de los efectos aquí programados son temporales. Esto quiere decir que muestras ya pasadas o incluso futuras son utilizadas para confeccionar una presente, cosa que nos lleva a almacenar muestras en registros. Estas muestras pasadas que volverán a ser necesarias en el futuro se almacenan en bloques de memoria o buffers. Este tema se puede tratar ampliamente, nosotros trabajaremos con un tipo de bloques de memoria que son los buffers circulares.

3.1.4.Buffers circulares

Son buffers que resultan útiles cuando se requiere almacenar *n* muestras pasadas o futuras. En el caso concreto que estamos viendo tratamos con un número *n* de muestras que ya han sonado, es decir, que pertenecen al pasado. Si no dispusiéramos de este tipo de buffers tendríamos que tener uno convencional con todas las muestras que tiene nuestra señal de entrada, en el caso de una señal de audio miles. La señal iría pasando e iríamos guardándola en los registros y cuando tuviéramos que hacer sonar una nota retrasada solo tendríamos que ir al registro donde se encuentra guardada. En este caso el código sería lento e ineficiente ya que necesitamos variables enormes que al procesador le cuesta manejar. La solución, como anticipamos, son los buffers circulares. En los delays necesitamos las muestras anteriores a la actual, que han sonado hace un tiempo menor o igual al tiempo de delay. Las anteriores a estas ya no las vamos a necesitar por tanto no es necesario que sigan ocupando memoria y ralentizando el código. Por lo tanto tan solo necesitamos un buffer que tenga tantos registros como las muestras correspondientes a un tiempo de delay.

Conforme entra la señal la vamos guardando en los registros del buffer circular y el puntero va avanzando. Cuando éste inserta el último registro, la próxima muestra será grabada en el primer registro, chafando así la muestra más antigua, que es anterior al delay con el que trabajamos, de modo que ya no interesa y puede perderse. Y así volvería a empezar el círculo.

Fig 3.1 Llenado inicial de un buffer de muestras

Siguiente paso, entra la muestra siguiente a la mn, la mn+1, Como ya no cabe al final, la metemos en el primer registro.

Fig 3.2 Acción de un buffer circular cuando queda lleno

3.2.Explicación funcional del programa

En este apartado se expone la funcionalidad del fichero que contiene la función principal *main*. Es la parte del programa que se encarga de la ejecución. En la parte inicial están declaradas las constantes que después serán utilizadas. La mayoría de ellas corresponden a las características de cada una de las familias de efectos. Algunas otras corresponden a aspectos del procesado como la cantidad de tiempo a procesar o el nombre y path del fichero que queremos procesar. Es aquí donde podremos cambiar, según queramos, las cualidades del efecto/os que van a ser utilizados. Podremos cambiar los valores de cantidad de retardo, amplificación de realimentación, frecuencia de osciladores, etc.

Una vez introducidas las características deseadas a las constantes tenemos que elegir los efectos por los que la señal deberá ser procesada. Podemos colocar o quitar efectos a nuestro antojo ya que gracias al procesado muestra a muestra los efectos se pueden ver como cajas negras donde entra una muestra y sale otra procesada, de modo que el sistema en general también lo podemos ver como otra caja que contiene cajitas más pequeñas. En el sistema siempre entrará una muestra y saldrá otra igual que en los efectos, independientemente del número de efectos que coloquemos, tan solo aumentaremos la cantidad de procesado. Para desactivar un efecto solo tendremos que comentarlo, o descomentarlo para activarlo. También podremos colocarlos a nuestro antojo cortándolos y pegándolos en el lugar que nos convenga.

La funciones de los efectos las podremos colocar en dos lugares distintos que llamaremos **etapas**. Por un lado está lo que he llamado **etapa de feedback general**, y por otro la **etapa serie**, que por su estructura podrán obtener resultados distintos. Es importante entender el porque de estas dos etapas así que están explicadas detalladamente a continuación.

Los efectos temporales aquí programados tienen unas características especiales que les permiten actuar de dos maneras distintas, utilizando buffers generales o comunes con otros efectos, y buffers propios e independientes de los demás efectos. Por esta razón se han programado dos etapas distintas que permitan el desarrollo de estas facetas.

3.2.1.Etapa Feedback general

Esta es la etapa que encontramos en primera posición. Como los efectos temporales utilizan muestras almacenadas a parte de la muestra que está siendo actualmente procesada, tenemos que obtener esas muestras de los buffers donde han sido almacenadas. En esta etapa utilizaremos un buffer común para todos los efectos, es decir, todos irán a buscar las muestras que sean necesarias al mismo buffer, de ahí el concepto de 'general'. La muestra de salida de esta etapa será almacenada en el lugar correspondiente de este buffer general y cuando uno de los efectos este siendo procesado irá a buscar las muestras que le hagan falta a este buffer mismo. De este modo todos los efectos serán muy dependientes del resto ya que todos estarán afectados de forma directo por los demás.

Para que las funciones vayan a buscar las muestras a este buffer general lo que haremos es enviarles como parámetro este buffer.

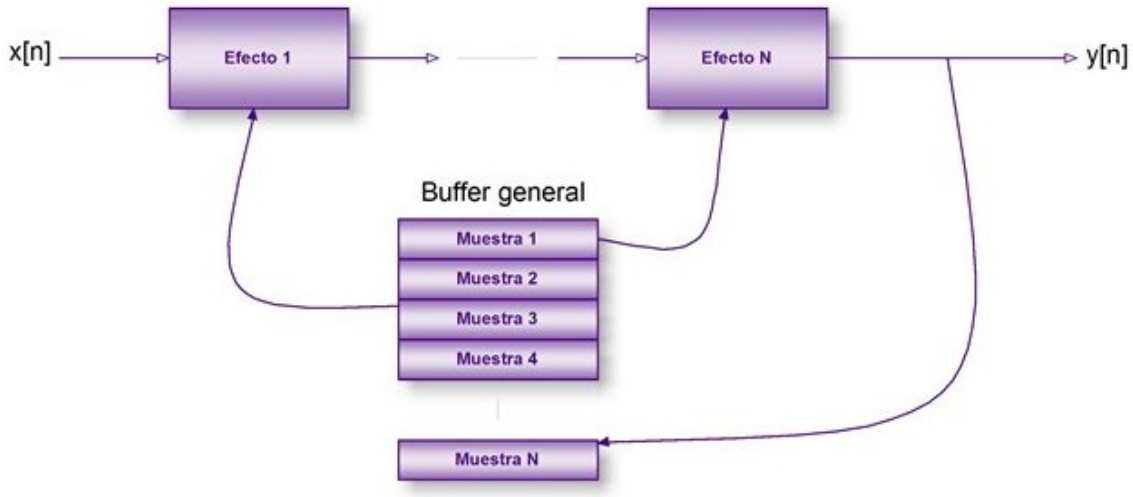


Fig 3.3 Esquema de la etapa serie

3.2.2.Etapa serie

Dado que esta etapa se encuentra en segunda y última posición la salida total de la etapa será también la del sistema. En esta etapa los efectos tendrán un buffer particular para cada uno en lugar de compartir uno. A cada efecto se le pasará su buffer correspondiente para que vaya a buscar la muestra que corresponda. En este caso los efectos no sufrirán tanta condición por parte del resto, tan solo el producido por el hecho de estar detrás de otros efectos ya que implica que la muestra de entrada que llega ya viene alterada. Sin embargo los efectos no sufrirán ninguna repercusión por parte de aquellos que se ejecuten después. Las salidas de cada efecto o subetapa serán almacenadas en estos buffers individuales.

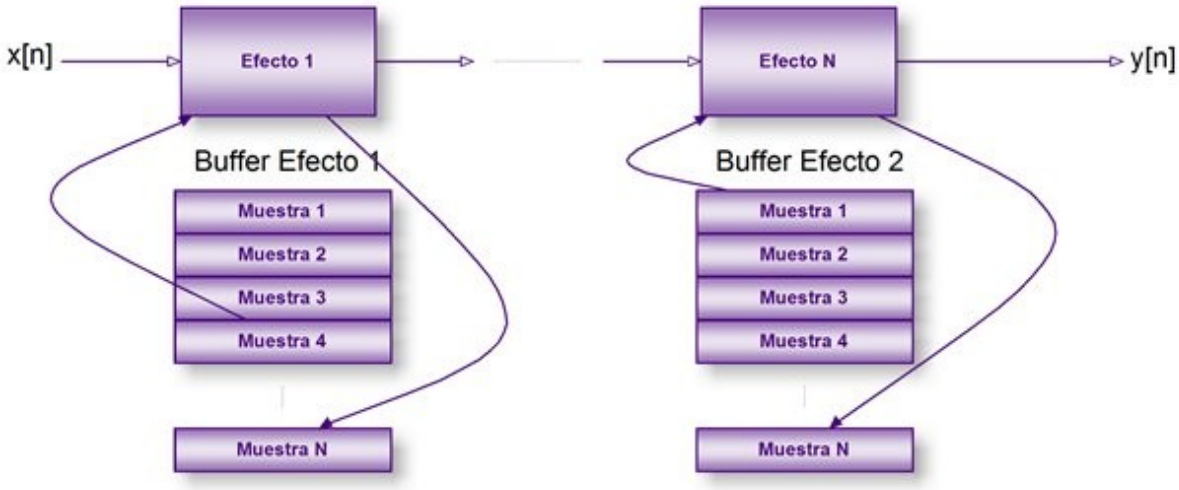


Fig 3.4 Esquema de la etapa de feedback general

3.3.Efectos de las etapas serie y feedback general

A continuación se detallarán los efectos utilizados en cada etapa. Se detallará un esquema y su ecuación en diferencias. La nomenclatura que se utilizará será x para la entrada, y para la salida, z para las muestras que vengan de fuera del efecto. También se hará una breve explicación del efecto que producen y de su implementación atendiendo a su diagrama.

Primero se expondrán los efectos de la etapa serie y después los correspondientes de la etapa de feedback general, y se dividirán por familias. Dado que la explicación e implementación de los efectos de la etapa de feedback general es muy parecida a la etapa serie en algunos casos esto se obviará. Tan solo será detallado en aquellos efectos donde se crea necesario.

3.3.1.Delays

3.3.1.1.Delay simple - etapa serie -

$$y[n] = x[n] + k*x[n - delay] \quad \textbf{(3.1)}$$

Efecto que produce

Este es el efecto de delay más básico que podemos construir. Lo que produce es una réplica de la señal que suena al cabo de un tiempo de delay. Lo podemos ver como un simple eco de una sola repetición. Es lo que provocaría el rebote de un sonido sobre una superficie. Podemos 'alejar' o 'acercar' esta superficie aumentando o reduciendo el tiempo de delay.

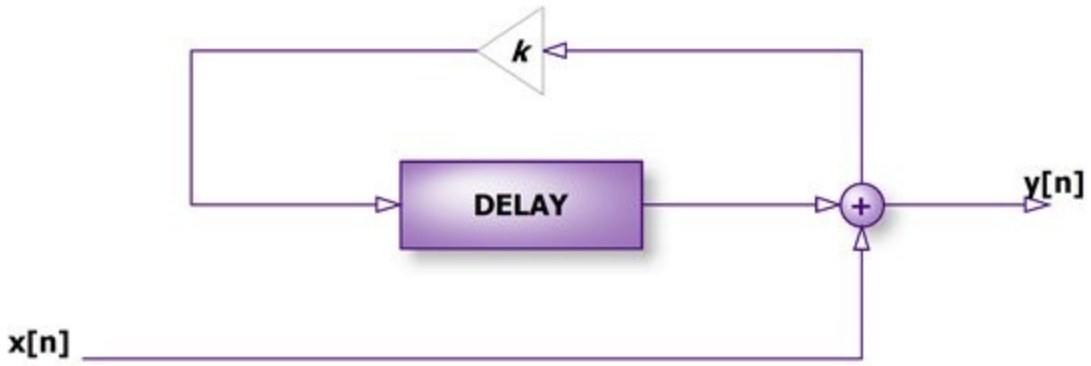
Implementación

La salida se obtiene sumándole a la muestra de entrada una muestra de entrada retrasada *delay* muestras, multiplicada por un factor *k*. El buffer que utilizamos almacena las muestras de entrada tal cual.

Como podemos observar colocamos amplificador a la salida de la muestra retrasada, por si queremos atenuarla o amplificarla a la hora de sumarla con la

señal actual. Será importante controlar el nivel de la señal retrasada teniendo en cuenta la importancia que se le quiere dar a esta.

3.3.1.2.Delay feedback - etapa serie -



$$y[n] = x[n] + k*y[n-delay] \quad \textbf{(3.2)}$$

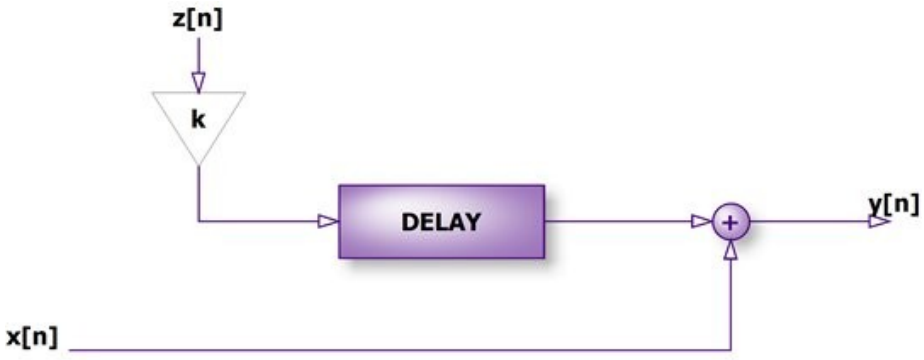
Efecto que produce

La diferencia básica con respecto al delay simple radica en que la réplica que producimos no es de la entrada si no de las salidas. Las salidas siempre volverán a sonar al cabo de un tiempo de delay, por lo que nunca dejarán de sonar. Lo que podemos hacer es ir atenuando estas salidas de modo que se vayan perdiendo paulatinamente, más rápida o más lentamente, dependiendo de nuestras necesidades.

Implementación

La salida se obtiene sumándole a la muestra de entrada una muestra de salida retrasada *delay* muestras atenuada en un factor *k*. El buffer que utilizamos almacena las muestras de salida.

3.3.1.3.Delay feedback - etapa feedback general -



$$y[n] = x[n] + k*z[n-delay] \quad \textbf{(3.3)}$$

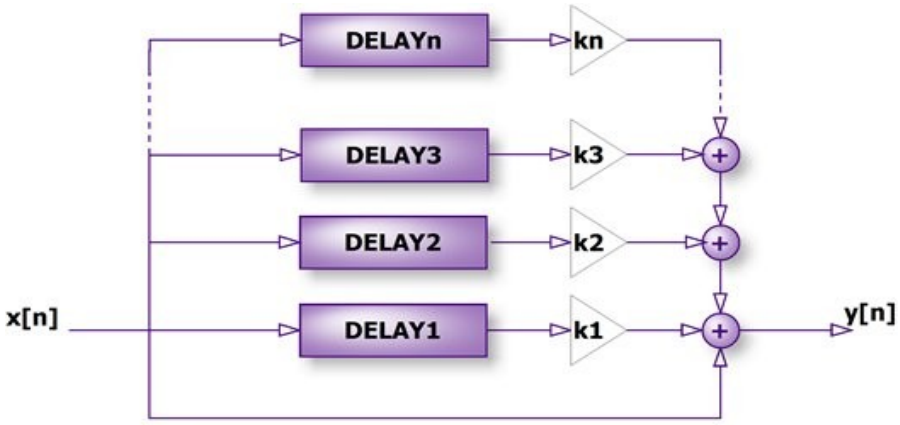
Efecto que produce

Difiere en el delay con feedback de la etapa serie en que salida con la que realimentamos el sistema viene de pasar por otros efectos por los que se verá afectado nuestro delay, cosa que no pasaba en la etapa serie.

Implementación

La salida se obtiene sumándole a la muestra de entrada una muestra de salida de la etapa de feedback general retrasada *delay* muestras atenuada en un factor *k*. El buffer que utilizamos almacena las muestras de salida de la etapa no del sistema del efecto que tratamos.

3.3.1.4.Delay multitap - etapa serie -



$$y[n] = x[n] + k1*x[n-delay1] + k2*x[n-delay2] + ... + kn*x[n\ delayn] \quad \textbf{(3.4)}$$

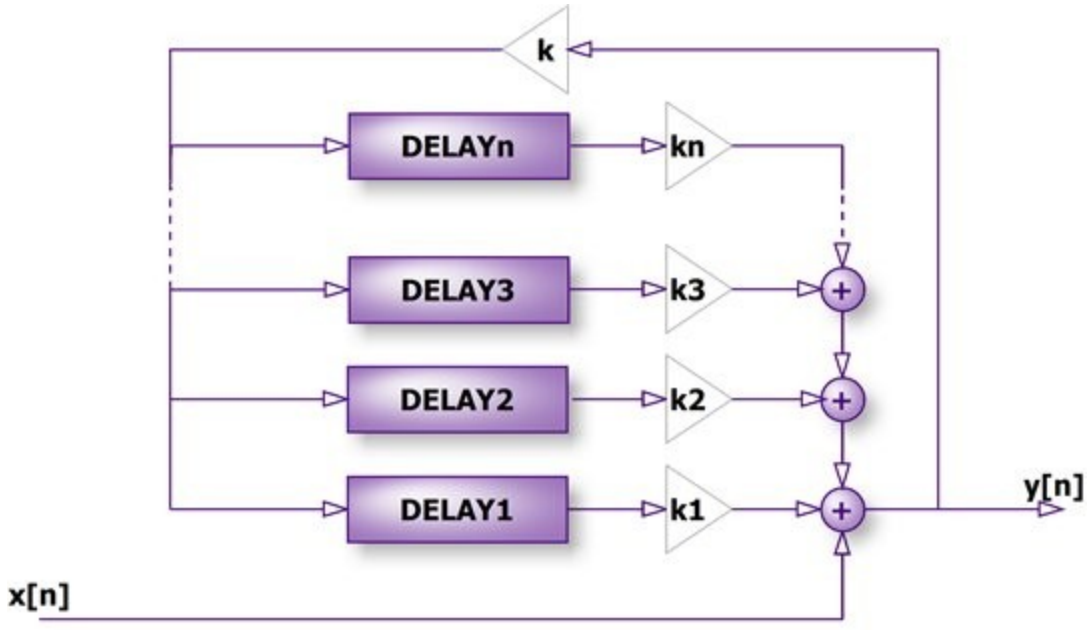
Efecto que produce

El efecto se explica de la misma manera que el delay simple. En este caso lo que sucede es que simulamos que en lugar de tener un superficie donde el sonido rebota tenemos varias que absorben una cierta cantidad de sonido y que están situadas a diferentes distancias y tan solo producen una reflexión cada una. Oiremos varios ecos de la señal a diferente volumen y con diferentes retardos.

Implementación

La salida se obtiene sumándole a la a la muestra de entrada muestras anteriores de entrada, cada una retrasada un determinado tiempo de delay. Todas las entradas retrasadas están atenuadas un cierto valor *kn.* En principio pueden haber tantas líneas de retardo como se quieran aunque cada vez el sistema adquiere más complejidad y volumen de procesado. En este trabajo fijamos en tres las líneas de retardo.

3.3.1.5.Delay multitap con feedback - etapa serie -



$$y[n] = x[n] + k*y[n-delay] + k1*y[n-delay1] + k2*y[n-delay2] + ... + kn*y[n-delayn] \quad \textbf{(3.5)}$$

Efecto que produce

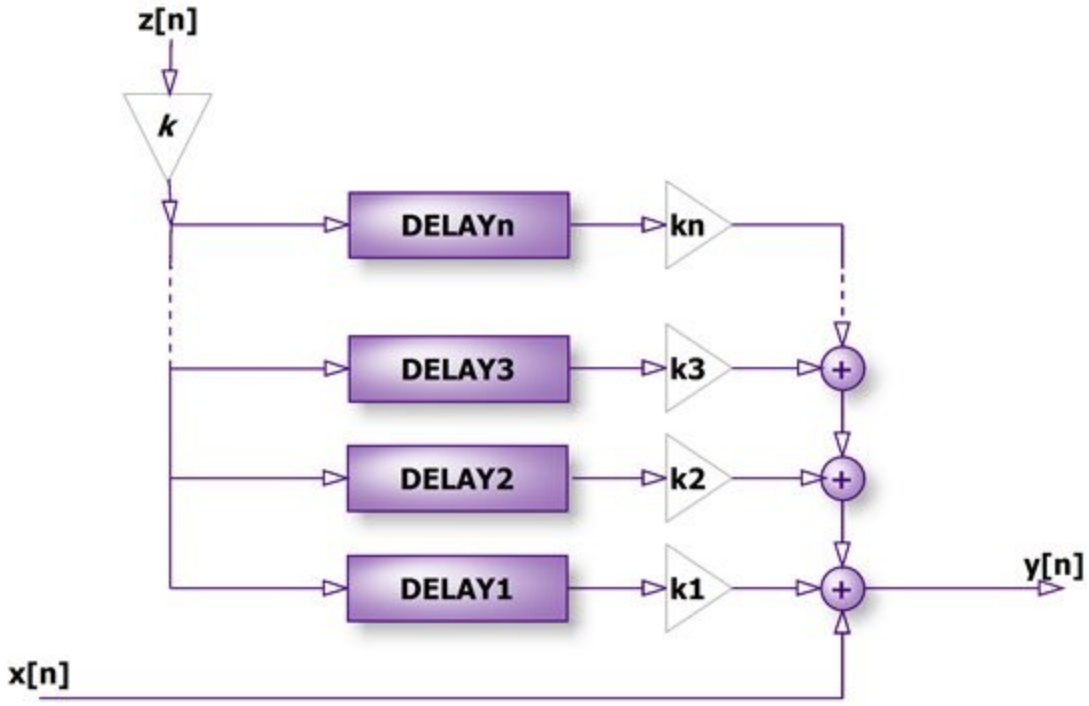
Al introducir feedback al delay multitap incrementamos mucho intensidad del efecto. Todos los rebotes de la señal se van repitiendo al cabo de un tiempo de delay. Se producen varias replicas de la señal de salida que vuelven a sonar siempre al cabo de cada tiempo de delay que determinemos para cada línea de

delay, y se irán perdiendo según el factor de atenuación que apliquemos a cada salida. Cuantas más etapas de delay pongamos más superficies de rebote estaremos simulando. De esta manera podríamos simular el situar la señal en una sala determinada, poniendo retardos suficientemente pequeños como para recrear el rebote del sonido sobre las paredes de esa confinamiento. Este efecto de recrear un sala recibe el nombre de *reverb*.

Implementación

La salida se obtiene sumándole a la a la muestra de entrada muestras correspondientes a salidas anteriores, cada una con un cierto retardo. Todas las salidas retrasadas están atenuadas un cierto valor *kn* y la salida que se utilizará para la realimentación también queda atenuada por un factor *k* para que resulte más sencillo controlarla.

3.3.1.6.Delay multitap con feedback - etapa feedback general -



y[n] = x[n] + k1*z[n-delay1] + k2*z[n-delay2] + ... + kn*z[n delayn] **(3.6)**

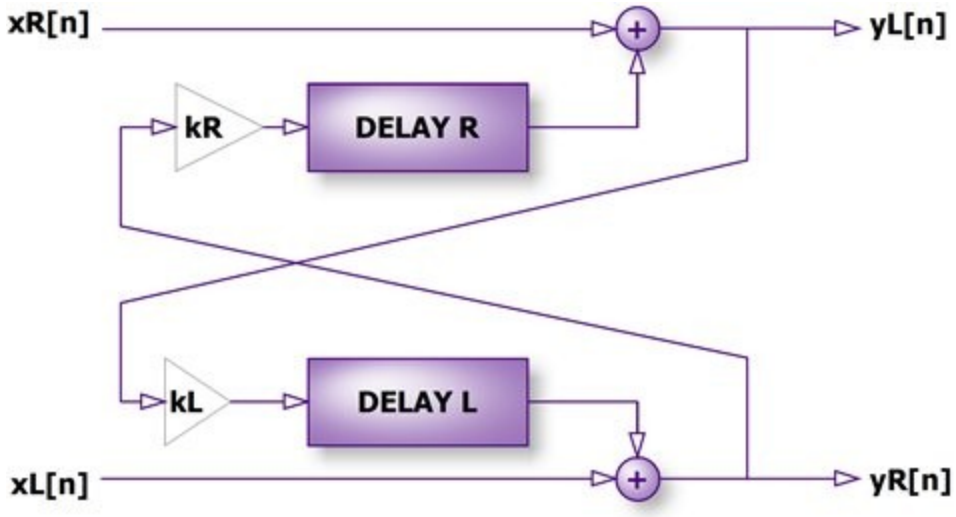
Efecto que produce

El efecto se vuelve a repetir tan solo con la diferencia de que al ser un buffer general el que nos provee las muestras estás también estarán condicionadas por todos los efectos posteriores a éste.

Implementación

La salida se obtiene sumándole a la a la muestra de entrada muestras correspondientes a salidas de la etapa de feedback general anteriores. Todas las salidas retrasadas están atenuadas un cierto valor *kn.* La realimentación que proviene de fuera del sistema, de la salida total de la etapa, queda atenuada por un factor *k*.

3.3.1.7.Delay de ping pong - etapa serie -



$$yL[n] = xL[n] + k*yR[n-delayL]$$
$$yR[n] = xR[n] + k*yL[n-delayR] \quad \textbf{(3.7)}$$

Efecto que produce

Este tipo de delay produce un efecto de rebote entre los dos canales, izquierdo y derecho. Lo que sale del canal izquierdo sonará después de un cierto tiempo de delay de nuevo (realimentación), por el canal derecho, sumado a este canal. Y viceversa lo que sale del canal derecho saldrá al cabo de un cierto retardo por el canal izquierdo. Es decir contamos con dos líneas de delay independientes, e incluso diferentes, para cada canal y las salidas realimentadas y cruzadas. Mediante un esquema lo veremos con más claridad.

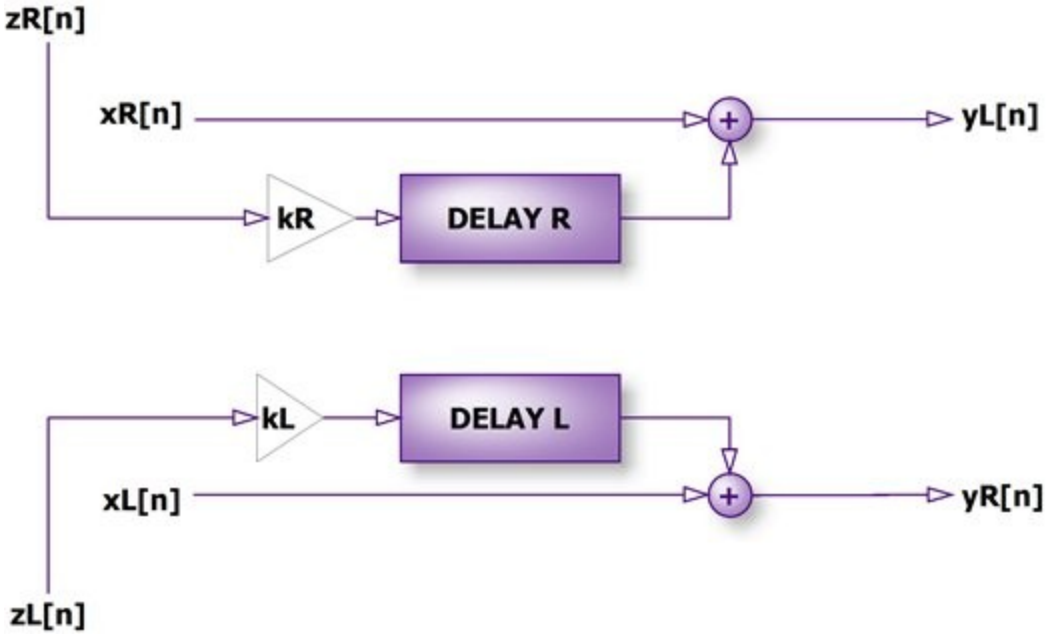
Implementación

Se trata de un efecto donde las salidas por el canal izquierdo o derecho no se tratan de la misma manera. Es por tanto un efecto que solo podemos apreciar si tenemos una señal estéreo. Su característica principal es que las salidas de los canales se cruzan y se suman a las entradas de cada canal. De este modo la salida del canal izquierdo se obtiene mediante la suma de la muestra de entrada del canal izquierdo y una salida del canal derecho anterior, producida

un cierto tiempo de delay antes. Y por el otro lado la muestra de salida del canal derecho se compone por la suma de la muestra actual de entrada del canal derecho más una salida del canal izquierdo un tiempo de retardo determinado anterior.

Tendremos un buffer para cada canal donde recoger todas las salidas que volverán a ser utilizadas. Es aquí donde observamos el cruce de canales. El buffer izquierdo almacenará las salidas del canal derecho y viceversa.

3.3.1.8.Delay de ping pong - etapa feedback general -



$$yL[n] = xL[n] + k*zR[n-delayL]$$
$$yR[n] = xR[n] + k*zL[n-delayL] \quad \textbf{(3.8)}$$

Efecto que produce

Produce el mismo efecto de rebote que el de la etapa serie pero con la afectación producida por los efectos posteriores colocados en la etapa.

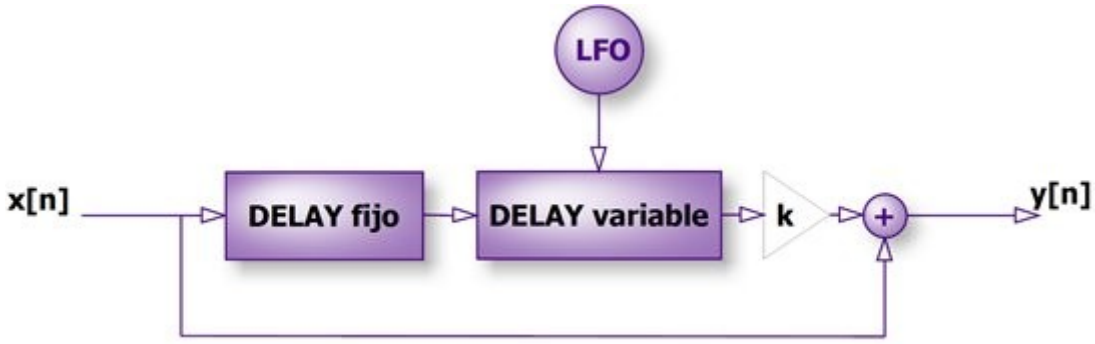
Implementación

La implementación resulta algo complicada ya que necesitamos dos buffers de almacenamiento, uno para las salidas del canal derecho y uno para las del canal izquierdo. Las salidas que tenemos que guardar son las totales de la etapa de feedback general por lo que no nos será suficiente con un solo buffer para almacenar estas muestras como hasta ahora. Tendremos que ir guardando ordenadamente las muestras de salida de la etapa según

corresponda dependiendo de si provienen del canal derecho o izquierdo. El funcionamiento será similar al del efecto en la etapa serie, tan solo que ahora cogiendo las muestras del buffer general, que en este caso estará dividido en dos, izquierdo y derecho.

3.3.2.Chorus

3.3.2.1.Chorus simple - Etapa serie -



$$y[n] = x[n] + x[n – (\text{delay Fijo} + (\text{delay variable}/2 * \text{LFO}[m] + \text{delay variable}/2))] \quad \textbf{(3.9)}$$

Efecto que produce

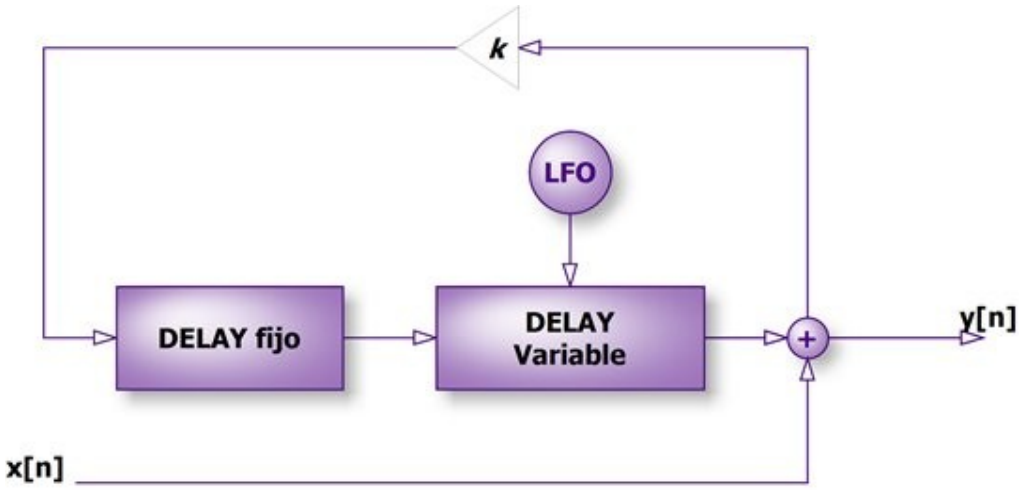
Produce un efecto de chorus sencillo donde tan solo tenemos dos instrumentos sonando a la vez, mientras uno suena con un retraso que va variando según un onda de algún tipo de baja frecuencia, y amplificada / atenuada con un factor *k.*

Implementación

Conformamos la salida con la muestra de entrada actual sumada a una muestra de salida anterior con un cierto retraso. Este retraso se mide con la suma de un tiempo de delay fijo de unos 20 ms más uno variable que va de 0 a unos 10 ms. Para determinar que valor tiene lo obtenemos de una onda periódica (seno, coseno, onda triangular, onda logarítmica, etc.) en un cierto punto controlado por un iterador, y lo multiplicamos por el valor máximo del delay variable. Comúnmente utilizaremos una onda periódica que oscile entre valores de –1 a 1. Si multiplicamos un delay por un valor negativo obtenemos un avance, no un retraso, cosa que no nos interesa. Para evitar esta situación el máximo valor de delay variable que utilizamos corresponde a la mitad del máximo deseado. La multiplicación por la onda correspondiente se hace con normalidad. En ese momento tendremos un delay oscilando entre delay

Variable/2 y –delay Variable/2. Si sumamos un valor de offset igual a delay variable/2 obtendremos el rango deseado entre 0 y delay variable/2.

3.3.2.2.Chorus con feedback - Etapa serie -



$$y[n] = x[n] + y[n - (\text{delay Fijo} + (\text{delay variable}/2 * LFO[m] + \text{delay variable}/2))] \quad \textbf{(3.10)}$$

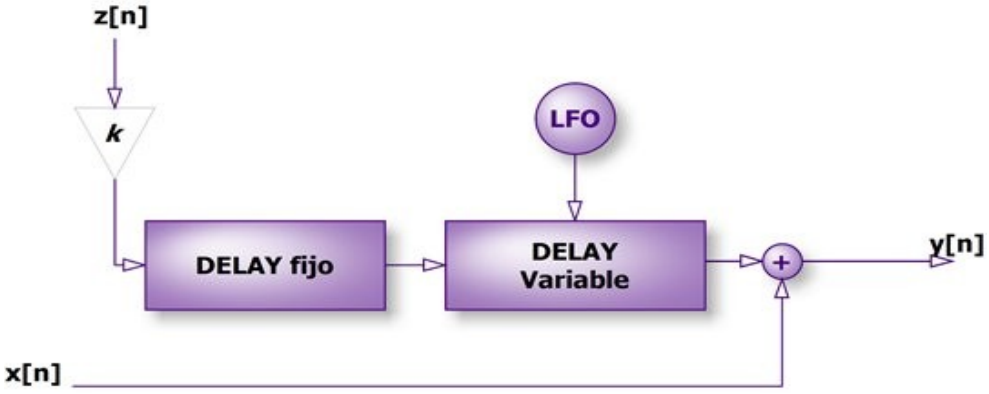
Implementación

Muy similar al chorus simple. Las muestras retrasadas se suman a la actual, pero en este caso no se almacenan muestras de entrada si no las salidas, he aquí la realimentación. El delay variable se calcula de la misma manera.

Efecto que produce

Obtenemos un efecto más intenso que sin la realimentación ya que la salida, que ya sufre efecto de chorus volverá a sonar un tiempo de delay más tarde. Se podría decir redundado que se va creando un chorus realimentado por una señal que ya de por sí está pasada por un chorus.

3.3.2.3.Chorus feedback - Etapa feedback general -



$$y[n] = x[n] + k*z[n - (delayFijo + LFO(m)*delayVariable] \quad \textbf{(3.11)}$$

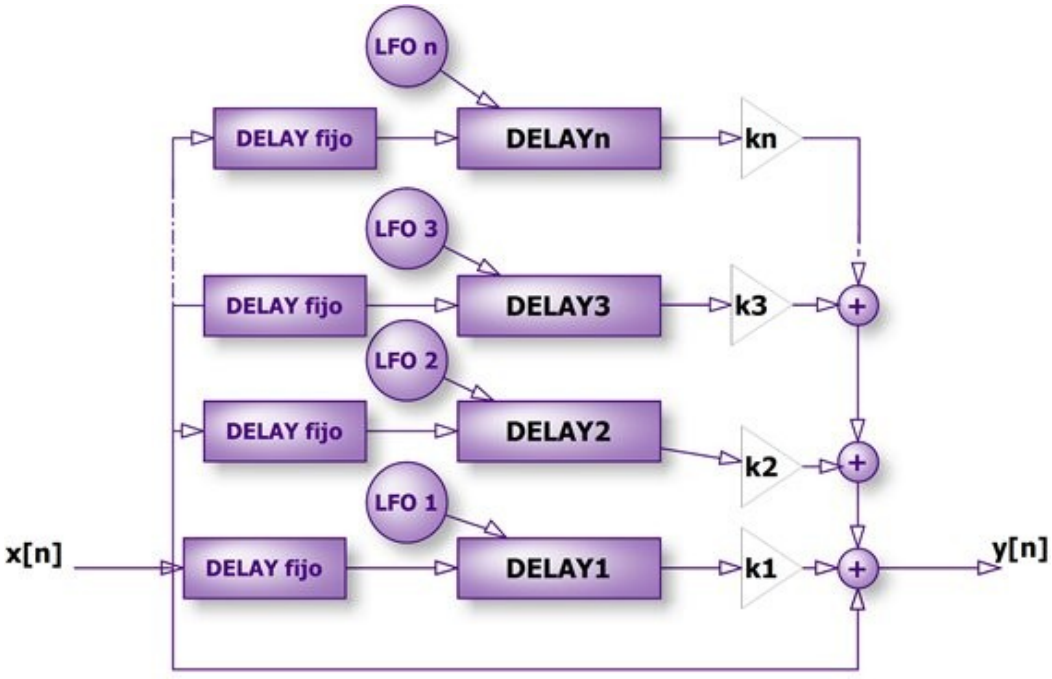
Implementación

En este caso el feedback viene de una señal externa al efecto que puede haber sido procesado por otros tantos y que portan vendrá más o menos afectada por estos.

Efecto que produce

Un chorus con feedback dependiente de las otras etapas que estén conectadas.

3.3.2.3.Chorus multitap - Etapa serie -



y[n] = x[n] + k*x[n – (delay Fijo + LFO(m)*delayVariable] + k1*x[n – (delay Fijo1 + LFO1(m)*delay Variable1] + k2*x[n – (delay Fijo2 + LFO2(m)*delay Variable2] + ... + k3*x[n – (delay Fijo3 + LFO3(m)*delay Variable3] **(3.12)**

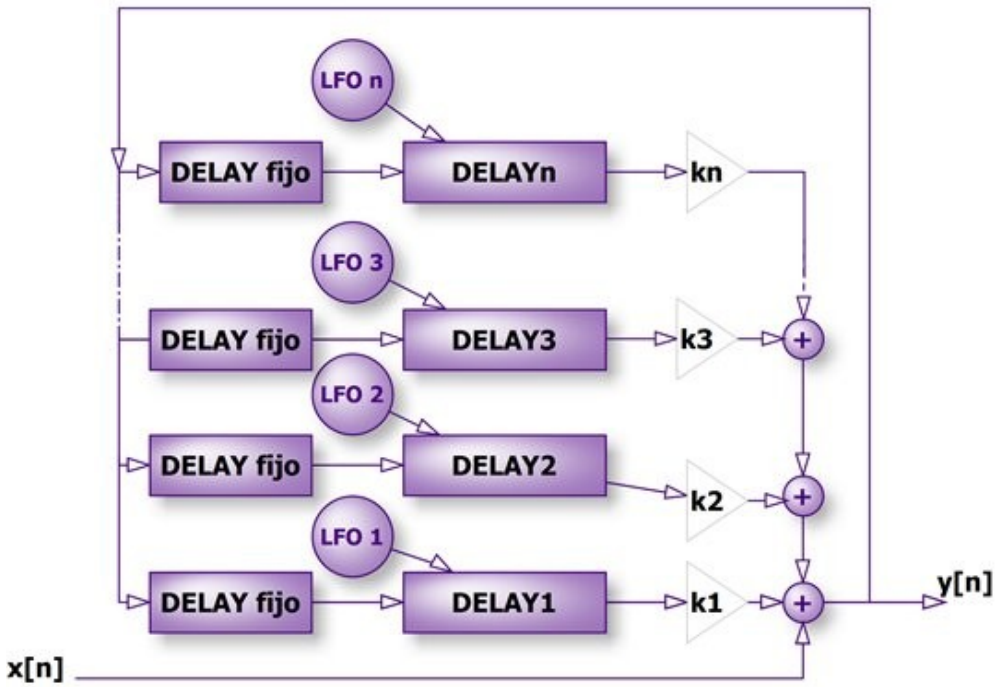
Implementación

En este caso lo que se pretende simular es un chorus creado por varios instrumentos, que le dará más intensidad. Se pueden introducir tantas etapas de delay como se quiera siempre atendiendo al volumen computacional. Cada delay tiene un valor y su propio LFO que hace que vaya variando. Cada uno tiene un control de volumen a la salida para darle más o menos relevancia.

Efecto que produce

Un efecto de chorus producido por varios instrumentos tocados al unísono. Se obtiene un sonido más complejo e intenso.

3.3.2.4.Chorus multitap con feedback (Etapa serie)



y[n] = x[n] + k*y[n – (delay Fijo +LFO1(m)*delay Variable1] + k1*y[n - (delay Fijo + LFO2(m)*delay Variable2] + k2*y[n - (delay Fijo + LFO3(m)*delay Variable3] + ... + kn*y[n – (delay Fijo + ondan(m)*delay VariableN] **(3.13)**

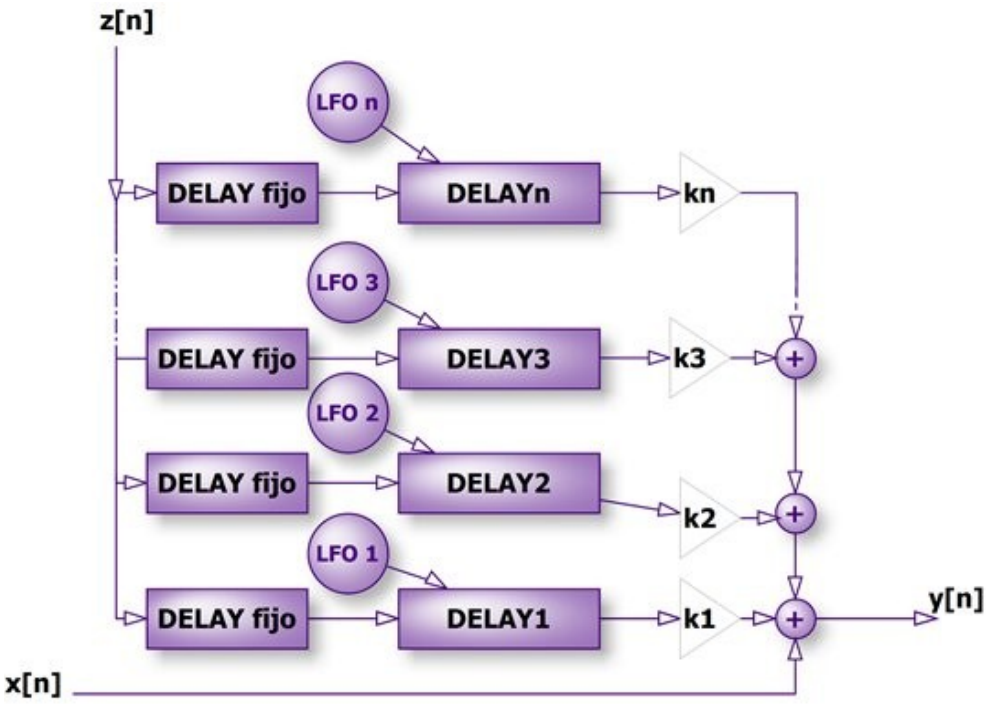
Efecto que produce

La realimentación produce más cambios en la fase de la señal, que se traduce en un chorus con matices algo distintos.

Implementación

La salida es realimentada y retrasada formando un sistema IIR.

3.3.2.5.Chorus multitap con feedback (Etapa feedback general)



y[n] = x[n] + k*z[n - (delayFijo + onda(m)*delayVariable] + k1*z[n - (delayFijo1 + onda1(m)*delayVariable1] + k2*z[n - (delayFijo2 + onda2(m)*delayVariable2] + ... + kn*z[n - (delayFijon + ondan(m)*delayVariablen] **(3.14)**

Implementación

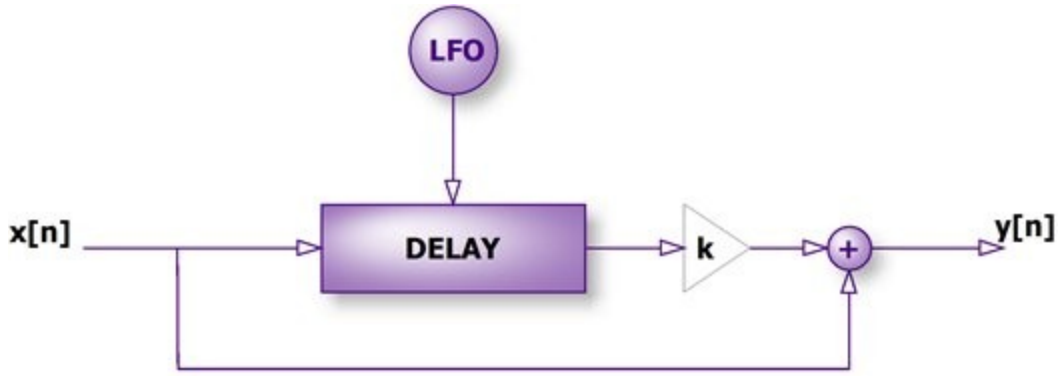
La realimentación viene de un buffer no dependiente únicamente del efecto en cuestión.

Efecto que produce

Siempre similar al que producen los efectos de chorus explicados anteriormente, con la dependencia que conlleva recibir la alimentación externa.

3.3.3.Flanger

3.3.3.1.Flanger simple - etapa serie –



$$y[n] = x[n] + x[n\text{-}delay] \quad \textbf{(3.15)}$$

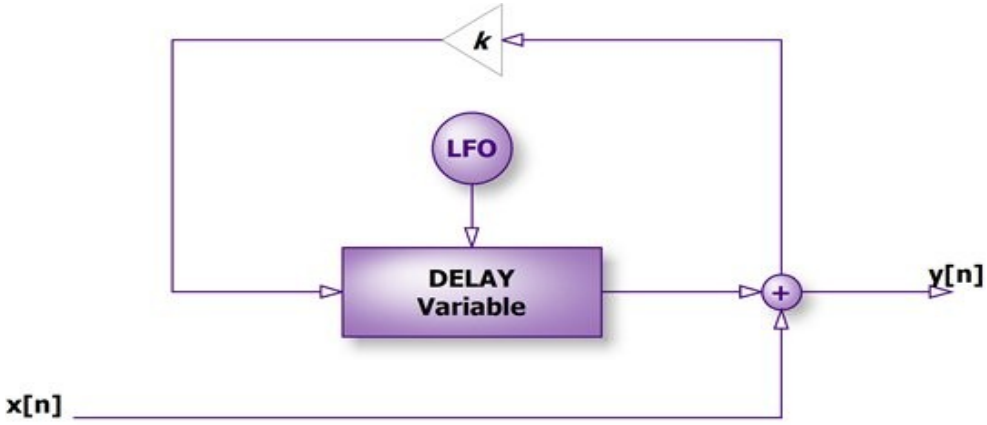
Implementación

La salida se obtiene sumando la entrada y una entrada retrasada, retraso que controla una línea de delay dependiente de un LFO. Para la obtención de un efecto de flanger es necesario que el retraso oscile entre 0 y 10 ms. El LFO en este caso será utilizado de manera similar al chorus, pero con alguna diferencia. Para obtener los valores de delay lo que haremos será tener un LFO que oscile entre (max_delay/2) y (-max_delay/2). Siempre sumaremos al valor que toque (max_delay/2) de modo que cuando esté en el mínimo esta suma resulte 0 (-max_delay/2 + max_delay/2 = 0) y cuando este en el máximo resulte el delay máximo (max_delay/2 + max_delay/2 = max_delay).

Efecto que produce

El efecto suave de flanger, que sobretodo se nota en señales con sonido muy constante.

3.3.3.2.Flanger con feedback - etapa serie –



$$y[n] = x[n] + y[n - delay] \quad \textbf{(3.16)}$$

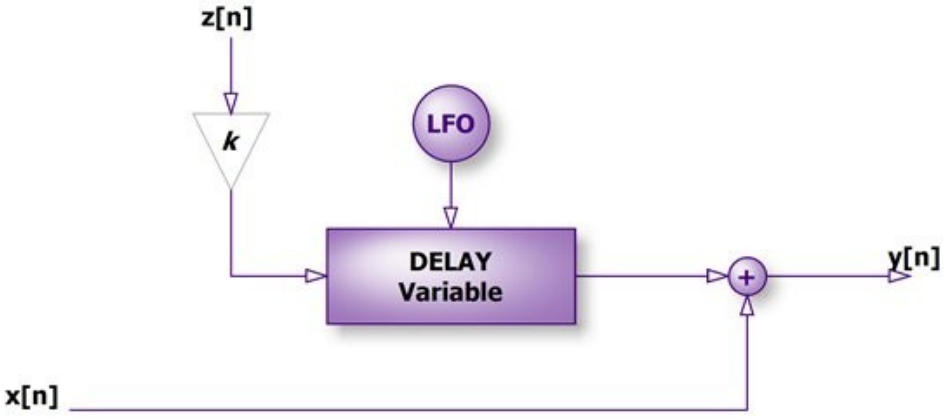
Implementación

Realimentamos el sistema con la salida.

Efecto que produce

Produce cambios de fase en la señal, y los ceros producidos a lo largo del eje frecuencial. Provoca un intenso efecto difícil de controlar.

3.3.3.3.Flanger con feedback - etapa feedback general –



$$y[n] = x[n] + z[n - delay] \quad \textbf{(3.17)}$$

Implementación

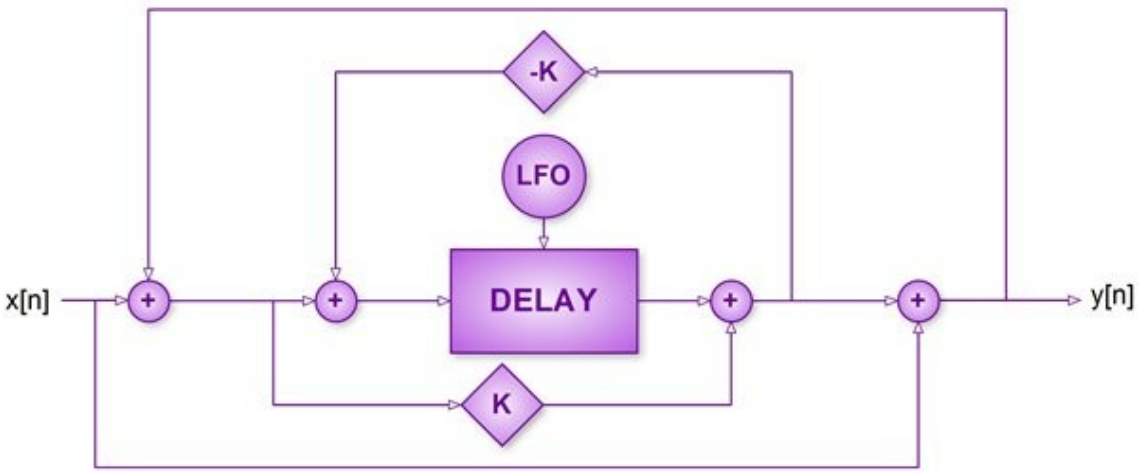
La realimentación proviene del exterior.

Efecto que produce

Como en todos los caso de un efecto de la etapa de feedback general, el efecto dependerá de el mismo y de todos los demás por los cuales está siendo afectada la señal. Difícil predicción.

3.3.4.Phaser

3.3.4.1.Phaser simple



$$y[n] = x[n] + K*(x[n] + y[n]) – K*(x[n\text{-}delay] + y[n\text{-}delay]) \quad \textbf{(3.18)}$$

Implementación

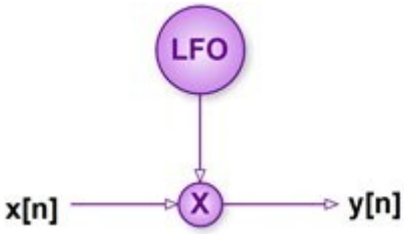
El sistema que encontramos es complicado de describir. Lo que encontramos es en medio un filtro pasa todo. La salida retrasada sirve para realimentar el sistema y es multiplicada por un factor -K, que a la vez le cambia el signo. La entrada actual sumada con la salida actual son multiplicadas por un factor K para obtener la salida total. Esta salida total es finalmente conformada con la entrada actual tal cual.

Efecto que produce

El efecto es complicado de describir. Es el provocado por la anulación de ciertas frecuencias que van cambiando y la manera de realizar esta anulación. Tiene un sonido entre un chorus y un flanger quizá.

3.3.5.Trémolo

3.3.5.1.Tremolo simple



$$y[n] = x[n]*LFO[m] \quad \textbf{(3.19)}$$

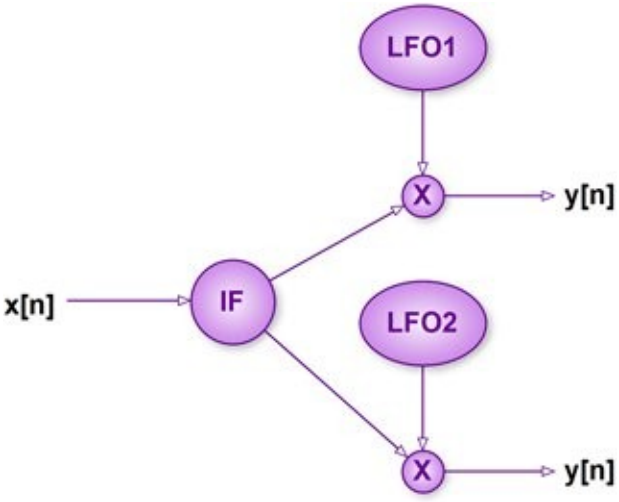
Implementación

La señal va siendo multiplicada por un onda senoidal que va tomando valores conforme avanza la señal. Tradicionalmente la frecuencia de la onda senoidal va acompasada con el tempo de la señal. De esta manera se producen caídas y subidas de volumen en la señal que van acompasadas, es decir que son un número entero en un compás. En el apartado dedicado al tremolo se explica como calcular que valor debe tomar la onda senoidal para adecuarla al tempo de la señal.

Efecto que produce

Produce una serie de subidas y bajadas de la señal tan frecuentes como sea el periodo de la onda senoidal utilizada. Se traduce un 'temblor' en la señal.

3.3.5.2.Tremolo doble



$$y[n] = x[n] * LFO(1 \text{ OR } 2) \quad \textbf{(3.20)}$$

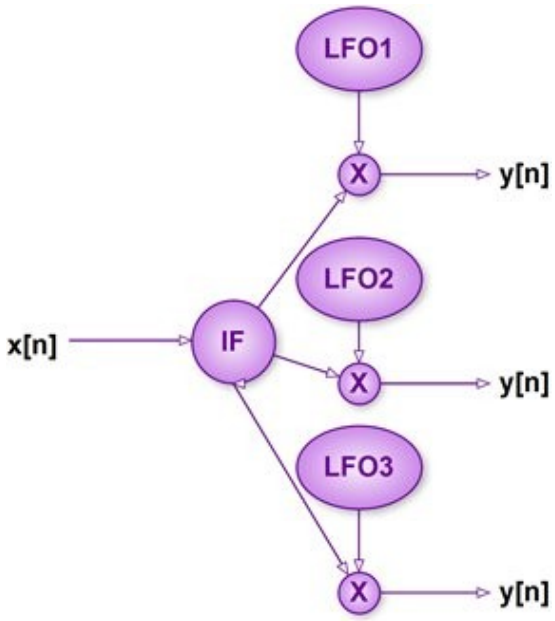
Implementación

En este caso se utilizan dos ondas senoidales de distinta frecuencia. Como ya hemos comentado se pueden hacer cálculos para acompasar el efecto al tempo de la señal.

Efecto que produce

Producirá dos temblores en la señal de distinta frecuencia.

3.3.5.3.Tremolo triple



$$y[n] = x[n] * LFO(1 \text{ OR } 2 \text{ OR } 3) \quad \textbf{(3.21)}$$

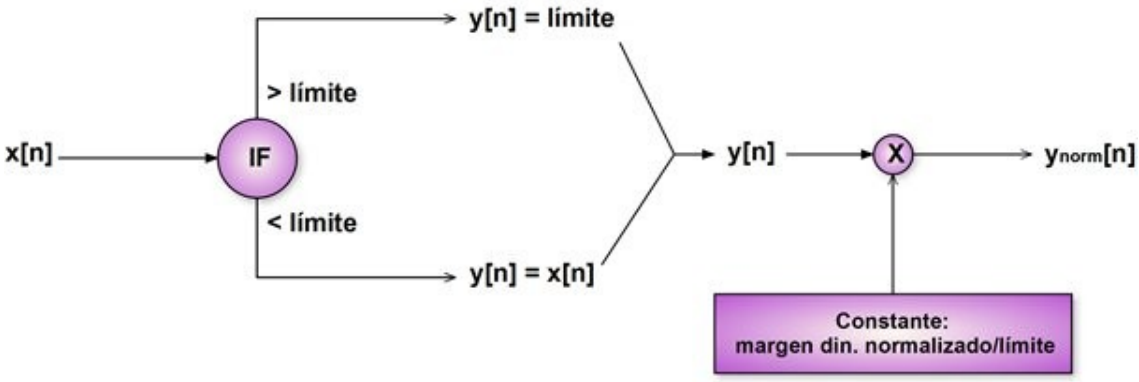
Implementación

Se utilizan tres ondas para producir el efecto de distinta frecuencia. Resulta más complejo el acompasar el efecto en este caso.

Efecto que produce

Se obtiene un sonido más complejo. Los temblores varían en tres ocasiones.

3.3.6.Distorsión y normalizador



$$y[n] = x[n] \text{ OR límite}$$
$$y_{norm}[n] = y[n] * (\text{margen din. normalizado / límite}) \quad \textbf{(3.22)}$$

Implementación

Se ha implementado una distorsión muy sencilla que consiste en reducir el margen dinámico de la señal. Para este propósito lo que hacemos es no dejar superar a la señal un cierto nivel. Cuando supera un cierto nivel como pasa cuando se supera un margen dinámico la señal siempre toma el mismo valor, que es precisamente ese valor máximo. Lo que nosotros podemos controlar es cual es ese límite, de modo que cuanto menor lo pongamos más reduciremos el margen dinámico de la señal y más distorsionada la encontraremos. Puesto que al reducir el margen también podemos reducir en gran medida el volumen de la señal a la salida de la distorsión situamos un normalizador de señal. Lo que hará esta etapa es una vez procesada la distorsión ampliar su volumen. Podremos elegir un nuevo margen dinámico y situar un nuevo máximo. Se traduce en que el límite que impusimos de la señal procesada por la distorsión pasará a ser el máximo situado por el normalizador. Para conseguirlo multiplicaremos cada muestra por el nuevo margen dinámico partido por el límite que habíamos situado para la distorsión. De esta manera cuando la muestra entrante tenga el valor de dicho límite el resultado de la operación será el valor del máximo del nuevo margen dinámico.

$$y[t] = x[t] * (\text{máximo del margen dinámico / límite de la distorsión}) \quad \textbf{(3.23)}$$

Puesto que trabajamos con muestras de 16 bits el margen dinámico de la señal varia entre $-2^{16}/2$ y $2^{16}/2$. De modo que no podremos indicar valores superiores a estos cuando queramos normalizar.

Una herramienta que puede servir para observar que valor podemos tomar para el límite de la distorsión es observar un gráfico temporal de la señal y que entre que rango del margen dinámico se mueve. Nos pueden ayudar las herramientas que contiene CCS para generar gráficos de las variables utilizadas.

Efecto que produce

Produce una distorsión mayor o menor de la señal según coloquemos un límite más grande o más pequeño. En el extremo, un límite tan grande como el margen dinámico no produciría ningún efecto sobre la señal, un límite igual a 0 produciría una distorsión total y aplanaría al 100% la señal.

3.4.Aspectos técnicos de la programación

3.4.1.Parámetros variables de los efectos

Los parámetros variables que caracterizan cada uno de los efectos están todos declarados como constantes. De esta manera tenemos definidas, al principio de programa ejecutable, todas estas constantes divididas según pertenezcan a una u otra familia de efectos o sean de carácter general, utilizadas por todos o más de una familia de efectos. Esta manera de orden y definición de los parámetros hacen que sea fácil acceder a ellos y poder cambiarlos a voluntad, ya que solo es necesario cambiar el valor de la constante para alterar de una manera u otra el efecto deseado. Todo está programado referido a estas constantes, sin ningún tipo de valor predeterminado en el código.

3.4.2.Tipos de variables y declaraciones

Una limitación que hemos impuesto al proyecto es que las señales a tratar debían tener muestras de 16 bits, por comodidad y sencillez a la hora de programar.

3.4.3.Estructura ‘wav_file’ y ‘cabecera’

Los tipos de variable wav_file y cabecera, son estructuras creadas especialmente para el programa. Contienen los aspectos que caracterizan un archivo wav. De esta manera tenemos almacenado el total de la señal a tratar y podemos ir procesándola muestra a muestra.

3.4.3.1.wav_file

```
struct wav_file
{
        short int num_channels;
        int sample_rate;
        short int bits_per_sample;
        char *data8; //si es de 8 bits
        short int *data16; //si es de 16 bits
        int n_muestras;
};
```

El puntero de 'short int' *data16* será el contenedor de las muestras de la señal. Está definido para albergar muestras de 16 bits, ya que una variable tipo short int es de 16 bits. El otro puntero, data8 queda en suspenso para albergar muestras de 8 bits en una futura ampliación.

En el int *sample_rate* guardaremos la frecuencia de muestreo de la señal que tratemos. Dado que un int tiene 32 bits seremos capaces de guardar

frecuencias de muestreo hasta **2^{32} = 4294967296**, que nunca alcanzaremos ya que se trata normalmente con frecuencias de muestreo no superiores a 48.000 Hz. No es necesario muestrear frecuencias tan altas, que superan con creces el espectro audible humano, que queda entre los 20Hz y los 20 Khz. Por la tanto según el teorema de Nyquist no sería necesario utilizar frecuencias superiores a 2*20 KHz = 44 KHz para no producir aliasing.

En *num_channels* guardaremos el número de canales que nos indique que tiene el archivo wav. El número de canales puede ser muy elevado pero nosotros trabajaremos tan solo con señales mono o stereo.

En *nbits* guardaremos el número de bits que tienen las muestras de la señal leída. Como ya adelantábamos tan solo trabajaremos con muestras de 16 bits pero el tener una variable donde guardemos este valor nos servirá para comprobar que realmente tiene esos 16 bits y para una futura continuación del proyecto donde se puedan elevar o reducir esta cantidad de bits.

3.4.3.2.cabecera

Es aquí donde almacenaremos las características del fichero wav importantes para su completa definición y posterior composición cuando tengamos la señal procesada. Aquí tenemos la definición:

```
struct wav_cabecera
{
        char chunkID[4];
        int chunkSize[1];
        char format[4];
        char subChunk1ID[4];
        int subChunk1Size[1];
        char audioFormat[2];
        short int numChannels[1];
        int sampleRate[1];
        char byteRate[4];
        char blockAlign[2];
        short int bitsPerSample[1];
        char subChunk2ID[4];
        int subChunk2Size[1];
};
```

Casi todos los campos entre el fichero de entrada y de salida se tiene que mapear 1:1, es decir sin cambiar nada. Así son los campos que se refieren al número de canales, número de bits por muestra, tipo de fichero... ya que se mantienen igual para el fichero de salida. No es así para los campos que se refieren a la longitud de la señal. Puede entrar una señal que tenga una duración de 10 minutos pero nosotros tenemos la posibilidad de tan solo procesar 10 segundos por ejemplo. En ese caso la duración de la señal de

salida difiere a la de la señal de entrada. El campo que se encarga de indicar cual es la cantidad de muestras que contiene la señal es el campo *chunkSize*. A este campo se le asigna el valor que se corresponda a la duración de la señal de salida. De otra forma cuando fuéramos a reproducir el archivo tendríamos errores por parte del reproductor.

3.4.4.Punteros y buffers

Los buffers están declarados como punteros. Apuntan a una zona de memoria libre. Para asegurar este hecho se parte de una zona de memoria que no utiliza el programa. Hay un fichero .cmd (linkador de memoria.cmd) asociado al proyecto que nos dice como esta ordenada la memoria. Los buffers dado que son variables que deben albergar un gran número de muestras tienen un tamaño muy elevado. La memoria que dispone la DSP va de la dirección 0x8000000 a la 8x1000000. De modo que tenemos 1.000.000 de bytes disponibles de memoria. La variable más grande que almacenaremos será la señal de entrada y de salida.

Una vez tenemos claro donde quedan declarados los buffers queda explicar de que manera aseguramos que nos se solapen entre ellos. Es un hecho a tener en cuenta cuando no se reserva explícitamente una cantidad de memoria, ya que lo único que indicamos en la declaración es el punto donde comienza el buffer no donde acaba. Para solventarlo debemos imponer algunas limitaciones.

3.5.Limitaciones

3.5.1.Limitaciones para buffers

En el caso de los buffers que almacenan muestras pasadas no tiene sentido que tengan unas dimensiones excesivas para lo que se espera de su uso. Por ejemplo en el caso del flanger los retrasos están como máximo sobre los 10 ms. En el caso de tener una frecuencia de muestreo de la señal de 22050 Hz los 10 ms corresponden a 22050 muestras/s * 10 ms = 220,5 muestras.

Por lo tanto no tendrá demasiado sentido tener un buffer que tenga una capacidad muy superior a 221 muestras, en bytes 221 muestras * 2 bytes/muestra = 442 bytes. Es decir del 1.000.000 de bytes disponibles deberíamos reservar estos 442 bytes si quisiéramos utilizar este flanger. En un ejemplo más concreto, imaginemos que disponemos de toda la memoria posible, es decir, de 0x80000000 a 0x81000000. Para hacer una reserva de 442 bytes podríamos colocar este buffer al principio del buffer apuntando a la primera dirección, la 0x80000000. El siguiente buffer que utilizáramos lo deberíamos colocar 442 direcciones más adelante, apuntando a la 8x0000443. Y así con el resto de buffers que utilizáramos.

En otro ejemplo imaginemos ahora que tenemos un señal mono y que queremos coger un tramo lo mayor posible de ésta. Nuestros buffers para los efectos ocupan 40.000 bytes. De modo que nos quedan 1.000.000 bytes – 40.000 bytes = 960.000 bytes disponibles. Tenemos que nuestra señal está muestreada a 22.050 Hz, tiene 16 bits (2 bytes) por muestra y 1 canal. Con la memoria disponible tenemos para 960.000 bytes / 2 bytes por muestra = 480.000 muestras. Para calcular el tiempo lo hacemos mediante la frecuencia de muestreo, 480.000 / 22.050 muestras por segundo = 21,76 segundos. Tendríamos la mitad de tiempo si trabajáramos con una señal estereo.

3.5.2.Tamaño de las señales a tratar

El hecho de que la memoria sea finita nos hace tener un tamaño máximo para las señales a tratar, tanto la de entrada como la de salida. Dado que las señales, pese a que sean de unos pocos segundos, tienen un tamaño muy considerable también son declaradas como punteros que apuntan a una dirección de memoria, con cuidado que tampoco se solapen con ninguno de los buffers. Aprovechando que una muestra de entrada corresponderá a una de salida, por el hecho de que el procesado es muestra a muestra, utilizaremos el mismo buffer para la señal de entrada que de salida. Cuando salga un muestra de entrada utilizaremos el espacio que ha dejado libre para insertar la muestra procesada de salida. De esta manera nos ahorramos el uso de un buffer de gran magnitud ya que corresponde a la señal de audio, buffers de mayor envergadura.

Para calcular la mayor cantidad de señal que podemos procesar debemos pasar la máxima capacidad de memoria de la DSP a muestras y

posteriormente a tiempo. Disponemos de un total de 1.000.000 de bytes. Cada muestra de las señales es de 16 bits lo que corresponde a 2 bytes. Contemos que las señales son stereo por lo tanto tenemos dos canales que suenan a la vez. 1.000.000 bytes / 2 bytes = 500.000 muestras; 500.000 muestras / 2 canales = 250.000 muestras por canal. De modo que contando con nuestra frecuencia de muestreo que es de 22050 Hz, podremos procesar una señal stereo de hasta 250.000 muestras / 22050 (muestras/segundo) = 11,33 segundos, 22,66 en caso de ser una señal mono. Si utilizáramos toda la capacidad para la variable que almacena la muestra de entrada y salida no dispondríamos de más memoria para los buffers de los efectos temporales, por lo que no podríamos utilizarlos, ya que sufriríamos invasiones de espacio de memoria. En todo caso las necesidades de memoria de estos buffers son muy pequeñas en comparación con la variable que almacena la entrada y salida.

3.5.3.Máxima capacidad de procesado en tiempo real

Dado que el procesado que se realiza es muestra a muestra y en tiempo real es muy importante saber la volumen de procesado que podemos hacer entre dos muestras correlativas. En las especificaciones técnicas de la DSP encontramos la frecuencia de reloj . Esta frecuencia es de 150 MHz, lo cual quiere decir que por cada segundo disponemos de 150 megaciclos de reloj. Para calcular cuantos ciclos de reloj disponemos entre muestra y muestra debemos fijar una frecuencia de muestreo de una señal, pongamos que ha sido muestreada a 22.050 Hz.

tiempo de muestra = 1 / 22050 Hz = 4,53 x 10e-5
tiempo de ciclo = 1 / 150 MHz = 6,67 x 10e-9
nº ciclos por muestra = tiempo muestra / tiempo de ciclo = 6791

Este resultado es para el caso particular de una señal muestreada a 22050 Hz. Conforme aumentemos esa frecuencia, por ejemplo al doble, el espacio entre muestra y muestra quedará reducido a la mitad, por lo que dispondremos también de la mitad de ciclos de reloj, menos tiempo para realizar procesado. El hecho de que la cantidad de ciclos disponibles sea finita nos lleva a tener limitaciones en cuanto al volumen de procesado, que tiene una relación muy directa con el número de efectos que podemos introducir.

Empíricamente, mediante el reloj que contiene el simulador de CCS, se ha calculado el número de ciclos que conlleva introducir cada efecto en una de las etapas; así como el número de ciclos que utilizan el resto de operaciones que se hacen para cada muestra. Los resultados son los siguientes.

Tabla 3.1 Relación de efectos y nº de ciclos que conllevan

Efecto	**Nº de ciclos**
delay simple	179
delay multitap	502
delay ping_pong	228
delay simple feedback	249
delay multitap feedback	628
delay ping pong feedback	298
flanger	793
chorus	471
chorus feedback	540
chorus multitap	1218
phaser	800
tremolo	792
tremolo doble	500
tremolo triple	684
limitador	79
flanger feedback	834
distorsión	82
normalizador dist	120
resto de operaciones	338

Para calcular el número máximo de ciclos en una vuelta de nuestro procesado debemos multiplicar por dos la suma de todos los efectos utilizados en ese momento y sumar el peso del resto de operaciones. Si queremos que el procesado sea posible en tiempo real el número de ciclos del procesado de una muestra debe ser menor al número de ciclos que disponemos para una frecuencia de muestreo de 22.050 Hz con muestras de 16 bits. El cálculo ha sido hecho en el capítulo 1, apartado 1.2 (Procesador digital de señal (DSP) TMS320C6711). El resultado era de 3399 ciclos para cada muestra tratada.

El número de ciclos utilizando todos los efectos es superior al número máximo de ciclos posibles para procesar en tiempo real. En nuestro caso, al no estar limitados por el tiempo podríamos utilizarlos todos pero para futuras ampliaciones ateniendo a la necesidad del procesado en tiempo real tendríamos que elegir los efectos deseados manteniendo un uso por de ciclos por debajo de los 3399 mencionados.

CAPÍTULO 4.CONCLUSIONES

De este trabajo he extraído sobretodo una gran carga de conocimiento alrededor de los efectos estudiados. De la mayor parte de ellos conocía básicamente su sonoridad, pero los a la hora de manipular los parámetros que los definen tan solo era una cuestión de prueba-error. Gracias a la investigación que he llevado a cabo tengo unas nociones mucho más sólidas y al utilizar pedales de efectos para guitarra, o instrumentos virtuales (plugins VST) soy capaz de predecir y obtener las sonoridades que deseo mucho más rápidamente y con conocimiento, no a costa de equivocaciones y pruebas.

He descubierto los algoritmos que desempeñan la mayoría de efectos, que son muy correosos dependiendo éstos de muchísimos factores. La mayor parte del tiempo dedicado ha sido para la compresión y programación de estos algoritmos. Sin embargo también es cierto que me gustaría haberle dedicado más tiempo al procesado y a la programación pero también me he encontrado con muchas trabas que saltar que ni siquiera eran propias de mi trabajo, como la lectura y composición de ficheros de audio. En este trabajo tan solo se han trabajado con ficheros wav, pero el abanico de tipos de archivos es muy amplio pasando por el mp3 o midi, que es el más utilizado en la composición con ordenador. He tenido que pasar muchas horas dedicadas a la programación averiguando los recovecos de los ficheros wav y la manera de interpretarlos. Otros handicaps como el uso de la DSP también han hecho mella en el tiempo dedicado propiamente al estudio y programación de los efectos. La memoria limitada de la DSP ha hecho que se tenga que ir muy con cuidado a la hora de realizar el procesado. La imposición sobre trabajar en tiempo real también ha sido un elemento a solventar y el propio hardware de la placa DSK en muchos casos un misterio. Quedan también, debido a estas incidencias, posibles puntos y aspectos a ampliar en un futuro.

Espero poder tener la oportunidad de, a medio plazo, poder continuar con este trabajo y acabar/continuar con las cosas que me quedaron pendientes, pero que no por ello son de menor interés para mi. En cualquier caso, el resultado que he obtenido ha cumplido con mis expectativas y objetivos marcados, y por ello quedo muy satisfecho por todo lo acontecido.

BIBLIOGRAFÍA

[1] Watkinson, J., “Advanced digital audio processing”, Cap. 4 en *The Art Of Digital Audio*, Focal Press Eds., pp. 85-139, Londres y Boston (1988).

[2] Arch, C.L., “Audio fundamentals”, Cap. 3 en *Principles of Digital Audio and Video*, Artech House, Inc., pp. 59-75, Londres y Boston (1997).

[3] Pohlman, K.C., “Digital signal processing”, Cap. 17 en *Principles Of Digital Audio*, McGraw Hill, pp. 685-734, USA (2005).

[4] Coulter, D., “Filtering”, Cap. 10 en *Digital Audio Processing*, CMP Books, pp. 241-250, Kansas (2000).

[5] Coulter, D., “Delay Effects”, Cap. 11 en *Digital Audio Processing*, CMP Books, pp. 259-275, Kansas (2000).

[6] Coulter, D., “Amplitude effects”, Cap. 12 en *Digital Audio Processing*, CMP Books, pp. 281-287, Kansas (2000).

[7] Coulter, D., “Combination effects”, Cap. 13 en *Digital Audio Processing*, CMP Books, pp. 291-299, Kansas (2000).

[8] Información sobre archivos wav *http://es.kioskea.net/audio/wav.php3*

[9] Información sobre archivos wav *http://en.wikipedia.org/wiki/WAV*

[10] Información sobre wav *http://www.borg.com/~jglatt/tech/wave.htm*

[11] Zölder, U. y Amatrian, X., *DAFX*, John Wiley and Sons Ed., Tottenham (2003).

[12] Teoría sobre efectos de audio digital *www.harmony-central.com*

[13] Chassing, R., *DSP Applications Using C, Wiley Ed. (Edición digital)*

Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTULO DEL TFC: Implementación de algoritmos de efectos de audio en un procesador DSP de TI

TITULACIÓN: Ingeniería Técnica de Telecomunicación, Especialidad en Sistemas de Telecomunicación

AUTOR: Cristian Quirante Catalán

DIRECTOR: Gabriel Montoro López

DATA: 06 de febrero de 2008

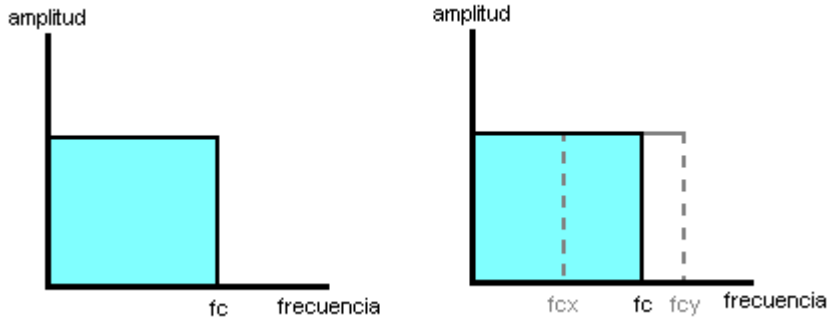
ANEXO I.EFECTOS DESCARTADOS

I.1.Filtro paso bajo dinámico

I.1.1.Explicación

Se trata de un filtro paso bajo sencillo como con el que cuentan las mesas de mezclas o cualquier software de edición de música digital. Se utiliza para enfatizar las partes zonas más graves del espectro frecuencial y atenuar las más agudas, controlando este hecho modificando la frecuencia de corte del filtro, u otros parámetros como la resonancia o la amplificación o atenuación de la banda de paso y la banda atenuada.

Lo más interesante es que crearemos un filtro paso bajo cuya frecuencia de corte no sea fija, sino que vaya cambiando automáticamente y se pueda caracterizar. La frecuencia de corte se moverá dentro de un rango que también será manipulable [fc máxima, fc mínima].



I.1.2.Variables

I.1.2.1.Las frecuencia de corte máxima y mínima.

Determinarán el rango desde donde y hasta donde se mueve la frecuencia de corte del filtro. Podremos hacer así que varie en un margen muy amplio y que filtre desde las más bajas a las más altas frecuencias, o que coja un pequeño rango y el efecto sea más marcado en esa zona en especial.

I.1.2.2.El número de coeficientes de filtro.

El número de coeficientes del filtro hará que la forma de onda de éste esté mejor o peor definida. A más número de coeficientes más cercana de la idealidad estará la forma de onda de nuestro filtro. Aumentar el número de coeficientes también implica aumentar el tiempo de procesado de modo que

tenemos que llegar a un punto de equilibrio entre la calidad que deseamos obtener y la complejidad / tiempo de procesado que podemos asumir.

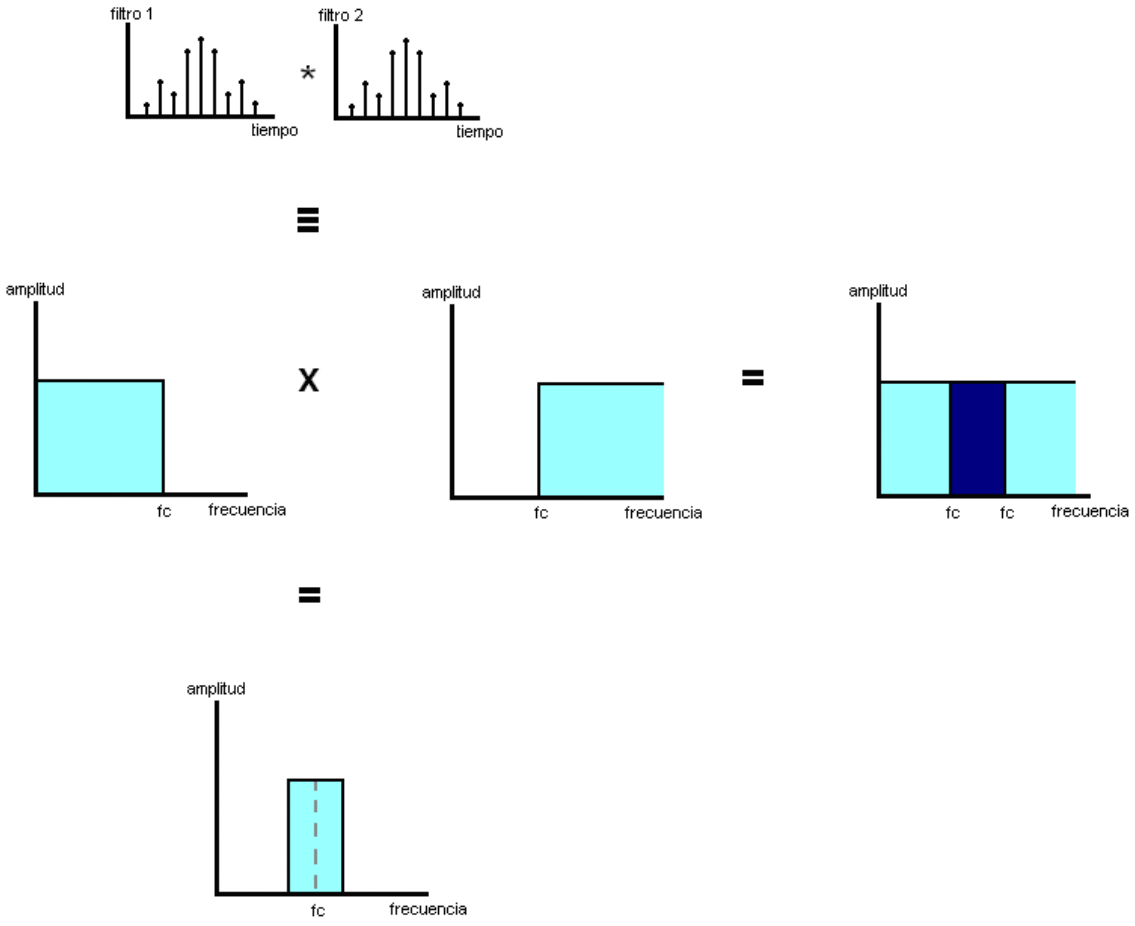
I.1.2.3.Velocidad de ascenso y descenso de la frecuencia de corte

Cambiando esta variable variaremos mucho el sonido producido por el efecto. Podemos hacer que el ascenso / descenso de la frecuencia de corte sea prácticamente instantaneo o que éste sea mucho más suave y lento.

I.2.Filtro paso banda dinámico

I.2.1.Explicación

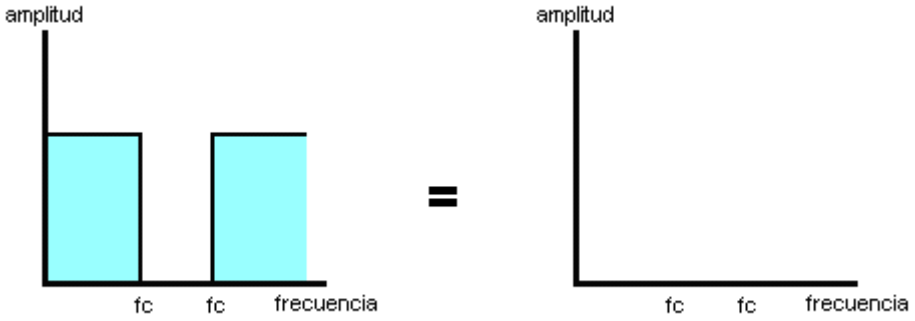
Para la obtención de un filtro paso banda haremos la convolución de un filtro paso bajo con uno paso alto. El ancho de banda del filtro resultante será más o menos grande en función de si las frecuencias de corte de los filtros paso alto y bajo son cercanas. Cuanto más cercanas sean menor será la banda que cubrirá el paso banda. Este hecho se produce ya que al convolucionar en tiempo lo que tenemos en frecuencia es una multiplicación. El efecto se puede apreciar más claramente en el dibujo:



Todas las bandas quedan atenuadas (idealmente hasta 0) menos las frecuencias que ambos filtros dejan pasar, paso alto y paso bajo, coinciden en dejar pasar. Esta banda de paso coincidente forma el filtro paso banda, cuya frecuencia de corte se encuentra en el punto medio de las frecuencias de corte de los filtro conformadores. Esta frecuencia se define como:

fc paso banda = fc paso alto + (fc paso bajo – fc paso alto)/2

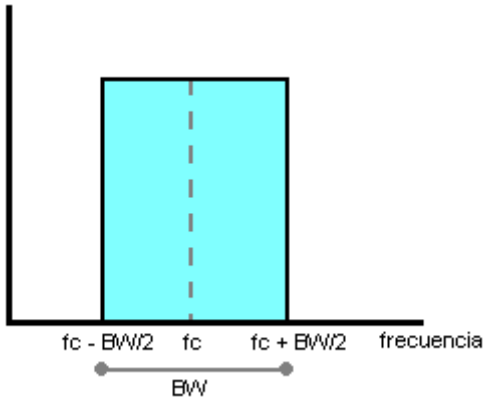
Obsérvese que este efecto no se produciría en caso de que la frecuencia de corte del filtro paso bajo fuera menor que la del filtro paso alto, en tal caso, toda la banda de frecuencias sería eliminada idealmente, muy atenuada en un caso real.



Para conformar el filtro en primer lugar tenemos que fijar el número de coeficientes que tendrán nuestros filtros paso alto y paso bajo. En segundo término debemos fijar un ancho de banda para nuestro filtro. Una vez hecho esto tendremos que programar los filtros paso bajo y paso alto de manera que dependan de la frecuencia de corte del paso banda. De esta manera las frecuencias de estos dos filtros tenemos que son:

$$f_{\text{filtro paso bajo}} = fc + BW/2$$
$$f_{\text{filtro paso alto}} = fc - BW/2$$

Se aprecia claramente en el dibujo:



Una vez calculados los coeficientes de los filtros conformadores deberemos realizar la convolución entre ellos para obtener finalmente el filtro deseado. Se debe tener en cuenta que el número de coeficientes de este filtro resultante será diferente al de los dos filtros conformadores. Recuérdese que tras una convolución de dos funciones, con n y m coeficientes respectivamente, el filtro resultante es uno con n+m-1 coeficientes.

I.2.2.Variables

I.2.2.1.Las frecuencia de corte máxima y mínima.

Sucede exactamente lo mismo que con el filtro paso bajo. Nos dictarán cual es el margen por donde se moverá la frecuencia del corte del filtro paso banda que en este caso se encuentra en la parte central de la banda de paso.

I.2.2.2.El número de coeficientes de filtro.

El número de coeficientes del filtro vendrá determinada como hemos explicado por los filtros paso bajo y alto conformadores. Como siempre a más cantidad de coeficientes, mayor calidad pero también mayor tiempo de procesado.

I.2.2.3.Velocidad de ascenso y descenso de la frecuencia de corte

Este tipo de filtro aumentará la presencia de una banda en concreto. Nosotros dictaremos cuanto tiempo debe quedarse en esa cierta banda controlando la velocidad de la frecuencia de corte.

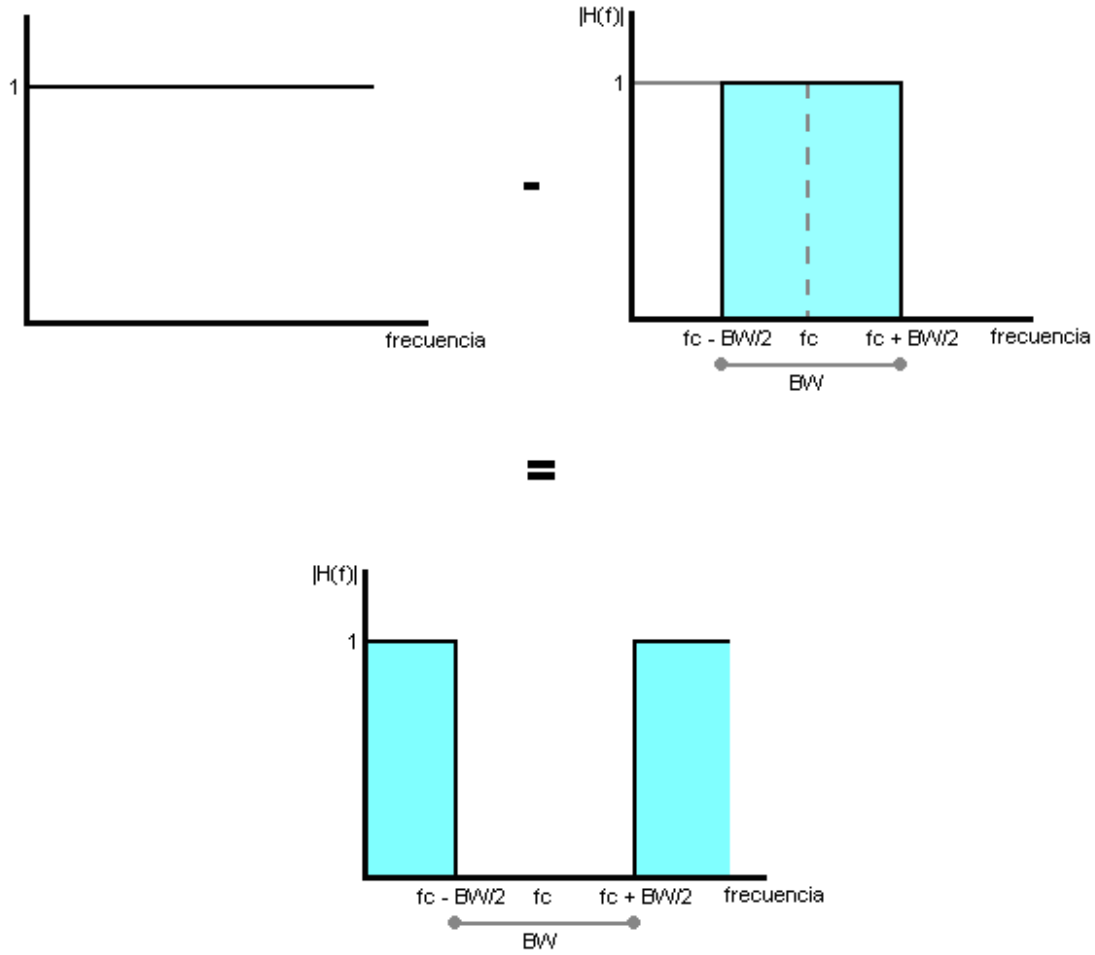
I.2.2.4.Ancho de banda del filtro

Este factor también será determinado por los filtros conformadores. Cuanto más cercanas estén las frecuencias de corte de estos filtros menor será la banda de paso del paso banda. Podemos hacer que sea más o menos estrecha según queramos dejar pasar una pequeña o extensa banda.

I.2.3.Otros usos

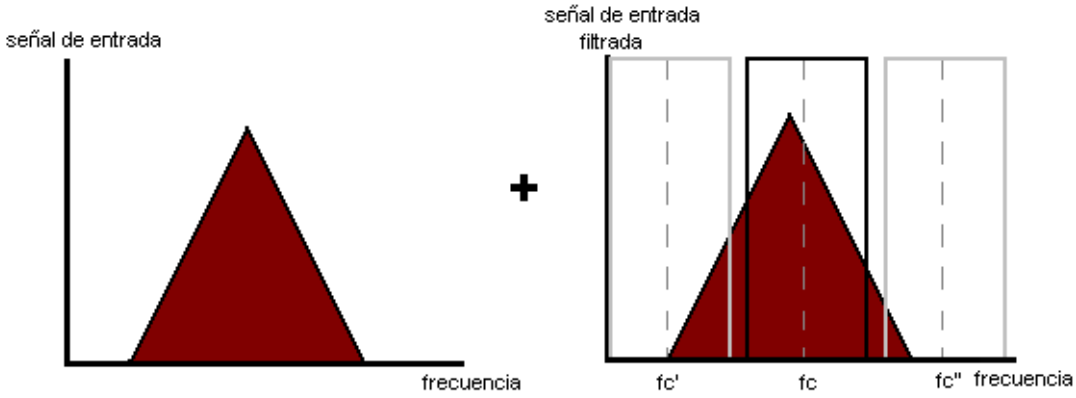
I.2.3.1.Filtro de banda eliminada

Una variante del filtro paso banda es el filtro que hace justo el efecto contrario. Es decir, filtra las frecuencias de una banda determinada. La obtención de esta función es muy sencilla una vez disponemos del filtro paso banda ya que su función será 1 – |H(filtro paso banda)|. Esta resta la encontramos en el dominio de la frecuencia y transformándola al dominio temporal tenemos que la operación a realizar es exactamente la misma 1 – |h(filtro paso banda)|.



I.2.3.2.Wah – Wah

El efecto de wah-wah lo introduzco dentro del apartado de filtro paso banda ya que básicamente funciona con un filtro de este tipo. Lo que provoca el característico sonido de este efecto es la suma de la señal de entrada tal cual con la misma señal filtrada con el filtro paso banda variando, en un tiempo determinado, su frecuencia de corte, aumentándola y reduciéndola. Normalmente este efecto es utilizado por guitarristas que controlan la frecuencia de corte mediante un pedal con el que presionando hacia delante aumentan la frecuencia de corte y hacia detrás la reducen, aunque no sean conscientes de ello...Dejando la frecuencia fija el efecto que se produce es el de amplificación de la banda donde cae dicha frecuencia.


señal de entrada
frecuencia
señal de entrada
filtrada
fc'
fc
fc"
frecuencia

ANEXO II.FUTURAS AMPLIACIONES

II.1.Configuración de los puertos de entrada/salida

A lo largo del trabajo se ha comentado el tipo de procesado llevado a cabo, muestra a muestra, enfocado a la implementación en tiempo real. No ha sido así a la hora de leer las señales de entrada a ser procesadas que se toman enteras. Una vez leídas y adoptadas como vectores de variables dentro del programa si que se van tomando las muestras una a una. El problema es que esta lectura no se hace en tiempo real y es algo lenta, es en este aspecto a lo que se debería tratar una futura ampliación haciendo capaz al sistema de recibir una señal de entrada y obtener al mismo tiempo una salida procesada de esta entrada.

La placa DSK con la que he trabajado dispone de un puerto de entrada con un conversor analógico-digital. Gracias a este conversor es posible conectar dispositivos que sacan audio analógico, como reproductores de CD, reproductores mp3... y obtener a partir de éstas muestras digitales capaces de ser tratadas mediante el programación realizada. También dispone de un puerto de salida con un conversor digital-analógico que una vez procesada la muestra es capaz de sacarla señal analógica, lista para ser reproducida por unos altavoces. La misma placa DSK dispone de un conector para altavoces. De modo que todo está integrado y dispuesto para que el procesado en tiempo real sea posible. Todos los procesos en el trabajo están enfocados en ese sentido, tan solo falta esta configuración de los puertos y conocer la manera de tratar la entrada y la salida de las muestras. Una vez se consiguiese esto el enlace con lo hasta ahora elaborado sería sencillo ya que todo esta dispuesto y preparado para ello.

II.2.Aumentar la compatibilidad con las señales a tratar

El trabajo estaba enfocado sobretodo en conocer y tratar con los efectos de audio digital. Durante la programación encontré un gran handicap a la hora de tomar las muestras de los archivos tipo wav de entrada. Estos ficheros pueden tener múltiples variables como pueden ser el número de bits por muestra, la frecuencia de muestreo, el número de canales, etc... a parte de la propia información de la forma de la señal. Son estas variables las que le dicen al reproductor de que manera debe interpretarlas para reproducirlas.

La dificultad es doble ya que la tenemos para realizar la lectura y en segundo término para construir el fichero de salida, listo para ser reproducido. Ya que no era mi máxima conocer la arquitectura de los ficheros wav decidí fijar un tipo de ficheros aceptados para realizar todo el proceso. La elección fueron los 22.050 Hz de frecuencia de muestreo, 16 bits por muestra y señales mono o estereo, en cuanto a canales. Estos son los requisitos de una señal para que pueda

superar todo el proceso diseñado. En todo caso el programa está preparado para operar con cualquier frecuencia de muestreo siendo capaz de crear un fichero plano para todas las diferentes a los 22.050 Hz fijados. Con este fichero plano se puede trabajar mediante alguna herramienta matemática como matlab, que permitiría reproducirla.

Una posible ampliación consistiría en hacer el programa más dinámico y compatible. Permitir el uso de señales de entrada de 8 bits, conseguir componer señales de salida con cualquier tipo de frecuencia de muestreo o dar la posibilidad de trabajar con señales de más de dos canales.

ANEXO III.ARCHIVOS WAV

Las señales de entrada a nuestro procesado son ficheros del tipo wav. Para poder tratarlas muestra a muestra debemos leer el fichero y extraer de él una serie de factores como pueden ser la frecuencia de muestreo, el número de bits por muestra, el número de canales y evidentemente también las muestras en sí que conforman la señal audible. Estos parámetros son los que dictan a los reproductores de que manera se deben interpretar esta señales para ser reproducidas. Las posibilidades son bastante grandes y se deben tener en cuenta.

La lectura del archivo consta de dos partes principalmente, por un lado la cabecera y por otro el contenido. La cabecera alberga todos los elementos relacionados con la conformación más técnica del archivo. El contenido contiene las muestras que formarán la forma de onda, que posteriormente se convertirá en sonido.

Las siguientes tablas muestran la disposición de los registros del fichero wav y sus posiciones de memoria relativas expresadas en bytes.

La primera tabla muestra los contenidos principales y a partir de esta se irán desglosando los grandes registros en partes más detalladas:

Dirección (byte)	Nombre	Tamaño (bytes)	Descripción
00h	rID	4	Palabra "RIFF"
04h	rLen	4	Tamaño del fragmento
08h	*rData	rLen	Datos del fragmento

*rData contiene la señal, codificado de la siguiente manera:

Dirección (byte)	Nombre	Tamaño (bytes)	Descripción
00h	wID	4	Palabra "WAVE"
04h	**Formato de la señal	18	Formato utilizado
1Ch	***Muestras de la señal	Dependiente de la la longitud de la señal	Datos

**Formato de la señal:

Dirección relativo (byte)	Nombre	Tamaño (bytes)	Descripción
00h	fId	4	Término "fmt " (el espacio es necesario)
04h	fLen	4	-
08h	*wFormatTag*	2	Formato (generalmente 1, para la modulación de código de pulso de Microsoft)
0Ah	nChannels	2	Número de canales (1=mono,

			2=estéreo)
0Ch	nSamplesPerSec	4	Frecuencia de muestreo (en Hz)
10h	*nAvgBytesPerSec*	4	nChannels * nSamplesPerSec * (nBitsPerSample/8) Para estimar el tamaño requerido por el buffer
14h	*nBlockAlign*	2	nChannels * (nBitsPerSample / 8) Para la alineación del bufer
16h	*FormatSpecific*	2	Número de bits por muestra (8 o 16)

***Muestras de la señal

Dirección (byte)	**Nombre**	**Tamaño (bytes)**	**Descripción**
00h	dId	4	Término "data"
04h	dLen	4	Longitud del campo dData (en bytes)
08h	*dData*	dLen	Muestras de sonido

El campo dData se formatea de la siguiente forma:

-En 8 bits mono: Cada byte representa una muestra

-En 8 bits estéreo: 2 bytes para cada muestra (canal izquierdo, canal derecho)

-En 16 bits mono cada palabra (byte bajo, byte alto) representa una muestra

-En 16 bits estéreo: 2 palabras para cada muestra (baja izquierda, alta izquierda, baja derecha, alta derecha)